Gobierno Bolivariano de Venezuela | Ministerio del Poder Popular para la Educación | Fundación Bolivariana de Informática y Telemática (Fundabit)

BICENTENARIO

# **Manual del Programador:** Recurso Digitalizado para el Aprendizaje "Sembrando para el Futuro".

**Desarrolladores:**

Homero Hernández
José Guerrero

# Introducción

El siguiente documento esta destinado a servir de guia la reutilización y/o extensión del ReDA “Sembrando para el futuro”, dirigido a programadores.

El Reda “Sembrando para el futuro” fue realizado en lenguaje Python versión 2.6, utilizando la librería python-pygame versión 1.8, la documentación de dicha librería puede ser consultada en la pagina web http://www.pygame.org/docs/ . Se utilizo el paradigma orientado a objetos diseñando las siguientes 8 clases: **boton, configuración, controlador, cursor, palabra, pantalla, interprete, vocabulario y texto**.

Cabe destacar que al momento de realizar la actual documentación el ReDA “Sembrando para el futuro” se encuentra en una etapa de desarrollo, lo que implica que la estructura y contenido de este manual estará sujeto a cambios.

Este documento tiene dos partes, la primera conformada por la descripcion de las clases antes mencionadas, y la segunda explicando la manera de expandir o editar el diccionario del intérprete virtual y su configuracion para ser visualizada desde el ReDA

# 1. Objetivo

Este documento tiene como objetivo describir los componentes del ReDA “Sembrando para el futuro”. Para cada componente se explica su función dentro del sistema, como esta constituido y algunos ejemplos sobre su uso.

# 2. Alcance

Este documento solo cubre la elaboración del ReDA “Sembando para el futuro” en el lenguaje de programación Python, utilizando la libreria python-pygame. No esta contemplada en el contenido de este manual, la modificación, adición o eliminación de componentes del sistema. Su proposito es orientar en la creación o modificación de Recursos Educativos siguiendo la estructura aqui descrita.

# 3. Componentes

El interprete virtual esta compuesto por los siguientes archivos:

- boton.py
- configuracion.py
- controlador.py
- cursor.py
- palabra.py
- pantalla.py
- texto.py
- interprete
- vocabulario

# 4. Dependencias

- **pygame:** Es un conjunto de librerias diseñadas para facilitar la creación de videojuegos o aplicaciones multimedia en Python. Para mas detalles consulte: http://www.pygame.org/news.html.
- **configparser:** Este modulo implementa un lenguaje basico para el analisis sintactico (parser) de archivos de configuración. Es necesario importar este modulo para leer y escribir parametros en el archivo de configuración **config.ini**. Para mas detalles consulte: http://docs.python.org/library/configparser.html.
- **subprocess:** Este modulo permite acceder a funcionalidades naitvas del sistema operativo para realizar tareas complejas o que requieren información del entorno en donde se esta ejecutando la aplicación. Es necesario importar este modulo para poder invocar el **interprete virtual**. Para mas detalles consulte: http://docs.python.org/library/subprocess.html.

# 5. Clases

## *boton*

### Descripción:

Esta clase contiene la información de los botones dentro de cada pantalla. Es el ente que se anima y recibe acciones.

### Estrucura(s) de datos:

- **identificador:** Representa el nombre del boton. Tipo de dato: **cadena**.
- **imagen:** Representa la imagen con todas las capturas de movimiento. Tipo de dato: **pygame.surface**.
- **divisiones:** Indica la cantidad de capturas de movimiento que tiene la imagen. Tipo de dato: **entero**.
- **fondo:** Representa la imagen de fondo. Tipo de dato: **pygame.surface**.
- **screen:** Representa la pantalla en la que se mostrara la animación. Tipo de dato: **pygame.surface**.
- **pos_x:** Representa la posicion del boton en la pantalla en el eje X. Tipo de dato: **entero**.
- **pos_y:** Representa la posicion del boton en la pantalla en el eje Y. Tipo de dato: **entero**.
- **estado:** Indica el estado de la animación del boton. Tipo de dato: **booleano**.
- **time:** Indica el tiempo de transicion de cada imagen en la animacion (solo aplica para imagenes dos o más divisiones). Tipo de dato: **entero**.

### Funciones:

**__init__():** Esta función se ejecuta al crear un objeto de la clase boton. Inicializa las variables, define el área de cada botón y la sucesión de imágenes que conformaran su animación. Como parámetros solicita:

- identificador.
- imagen.
- divisiones.
- fondo.
- screen.
- pos_x.
- pos_y.

**Ejemplo:**

>>> boton_acc = animacion(**“acc”, acc, 2, banner_inferior, pantalla, 60, 150**)

Donde **“acc”** es el identificador del boton, **acc** es la superficie que contiene la imagen del boton, **2** es el numero de fotogramas de la imagen, **banner_inferior** es la superficie sobre la que se muestra el boton, **pantalla** es la superficie de fondo, **60** y **150** representan la posición del boton en X e Y.

**animar():** Método que reproduce la animación del botón.

**update():** Método que verifica el estado del botón.

## *configuracion*

### Descripción:

Esta clase tiene como finalidad almacenar las variables de la configuración de accesibilidad del ReDA.

### Estrucura(s) de datos:

- **ind_pantalla:** Almacena el numero de la pantalla que se esta mostrando. Puede tomar valores entre 0 y 7. Tipo de dato: **entero**.
- **disc_audi:** Si es verdadera (True) se activan las caracteristicas del interprete virtual, si es falsa (False) se utiliza la configuración inicial. Tipo de dato: **booleano**.

### Funciones:

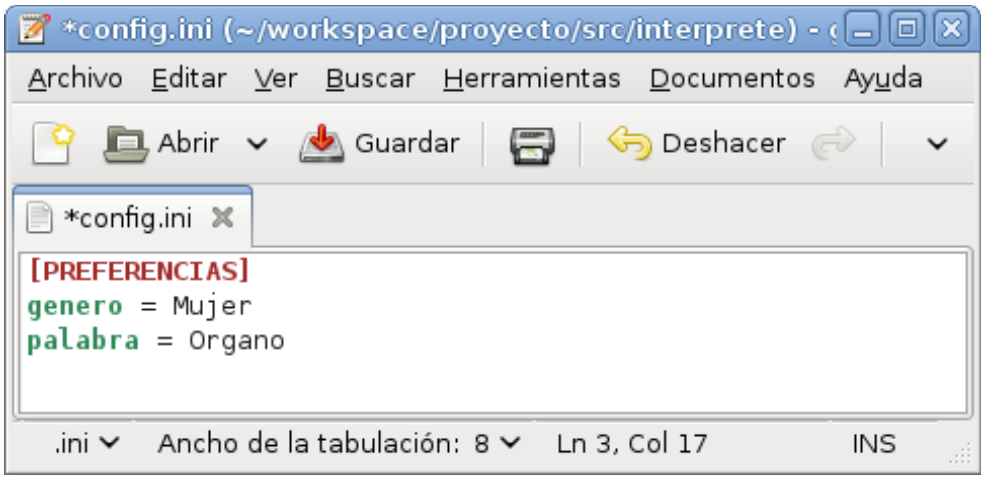
**__init__():** Esta función se ejecuta al crear un objeto de la clase configuracion. Iniciliza el indice de la pantalla.

**escribir_fichero():** Esta función sirve de interfaz para leer y escribir el archivo de configuración **config.ini**, guardando en este los parametros que seran utilizados por el interprete virtual (genero y palabra).

**Ejemplo:**

>>> escribir_fichero( “palabra”, “Organo”)
>>> escribir_fichero (“genero”, “Mujer”)

El archivo de configuración contendra los siguientes valores:



## *controlador*

### Descripción:

Esta es la clase mas importante del recurso. Contiene la logica para responder ante las acciones del usuario, hace llamadas a otras clases, crea y elimina instancias de otras clases.

### Funciones:

**__init__():** Esta función se ejecuta al crear un objeto de la clase controlador. Contiene el loop principal del programa. Recibe como parametro una pantalla que sera la interfaz grafica que mostrara al usuario.

**Ejemplo:**

>>> c1 = controlador(**pantalla1**)

Donde **pantalla1** es la pantalla que sera manejada por el controlador en su loop principal.

## *cursor*

### Descripción:

Esta clase identifica la posición del cursor y define la forma de su puntero. Esta clase hereda de **pygame.sprite.Sprite()**.

### Funciones:

**__init__():** Esta función se ejecuta al crear un objeto de la clase cursor. Recibe como parametro un entero que define la forma del cursor.

**Ejemplo:**

>>> raton = cursor(0)

Muestra el cursor normal.

>>> raton = cursor(1)

Muestra un cursor con forma de rombo.

>>> raton = cursor(2)

Muestra un cursor “roto”.

**update()**: Actualiza la posicion del cursor en la pantalla.

## *palabra*

### Descripción:

Esta clase define los metodos para crear palabras que seran utilizadas posteriormente como textos.

### Estrucura(s) de datos:

- **palabra:** Representa cada una de las palabras que contendra un texto. Tipo de dato: **cadena**.
- **interpretable:** Determina si una palabra podra ser explicada a traves del interprete virtual o el glosario de terminos. Si su valor es cierto (True) la palabra puede ser explicada y se resalta subrayandola. Si su valor es falso (False) la palabra no podra ser explicada y no sera resaltada. Tipo de dato: **booleano**.
- **medida:** Representa el tamaño de la palabra. Tipo de dato: **entero**.

### Funciones:

**__init__():** Esta función se ejecuta al crear un objeto de la clase palabra. Asigna una fuente tipografica, tamaño y verifica si la palabra es interpretable. Recibe como parametros:

- palabra.
- medida.

**Ejemplo:**

>>> palabra_n = texto(**“canción”, 16**)

Donde **“canción”** es el texto de la palabra y **16** es el tamaño de la tipografia.

## *pantalla*

### Descripción:

Representa la interfaz gráfica, contiene la configuración inicial de todas las pantallas.

### Estrucura(s) de datos:

- **x**: Representa el ancho de la resolución. Tipo de dato: **entero**.
- **y**: Representa el largo de la resolución. Tipo de dato: **entero**.
- **fondo**: Representa la imagen que se usara de fondo de pantalla. Tipo de dato: **pygame.surface**.
- **ubicacion**: Representa la posición en que se ubicara el fondo. Tipo de dato: **tupla**.
- **pantalla**: Representa la ventana donde se mostrará las imágenes y botones. Tipo de dato: **pygame.surface**.
- **banner_superio**r: Representa la imagen que se usara como banner superior. Tipo de dato: **pygame.surface**.

- **banner_inferior**: Representa la imagen que se usara como banner inferior. Tipo de dato: **pygame.surface** ó **nulo**.
- **grupo_palabras**: Representa un grupo de Sprites con las palabras. Tipo de dato: **pygame.sprite.Group**.
- **grupo_botones**: Representa un grupo de Sprites con los botones. Tipo de dato: **pygame.sprite.Group**.
- **grupo_anim**: Representa un grupo de Sprites con las animaciones. Tipo de dato: **pygame.sprite.Group**.

## Funciones:

**__init__()**: Esta función se ejecuta al crear un objeto de la clase pantalla. Recibe como parametro un entero que representa el indice de la pantalla.

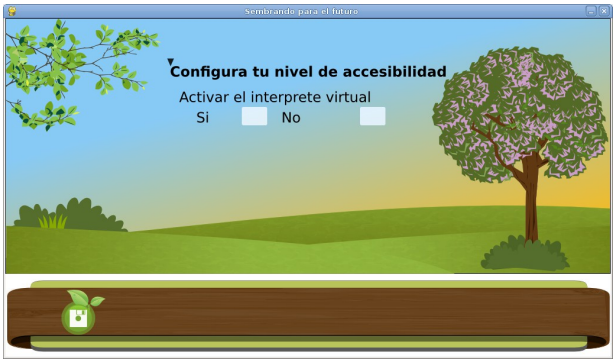
**Ejemplo:**

>>> fondo = pantalla(2)

Crea una pantalla utilizando el método como constructor. Muestra el menu:


>>> pantalla.__init__(1)

Crea una pantalla utilizando el método como función. Muestra la pantalla de configuracion:




**set_fondo()** : Muestra el fondo de pantalla y los banners superior e inferior según los valores asignados a los atributos: **fondo**, **banner_superior** y **banner_inferior**.

**Ejemplo:**

```
>>> pantalla.banner_superior =  pygame.image.load(“banerup.png”)
>>> pantalla.banner_inferior =  pygame.image.load(“banerdown.png”)
>>> pantalla.fondo = pygame.image.load(“casa.png”)
>>> pantalla.set_fondo()
```

**update():** Invoca el método **update()** de las clases botones y animaciones. Tambien invoca el método **set_fondo()**.

## *texto*

### Descripción:

Clase que se encarga de calcular la posición de cada palabra en la pantalla con un texto dado. Depende directamente de la clase palabra.

### Estrucura(s) de datos:

- **img_palabras:** Representa una lista de **sprites** creada para cada palabra. Tipo de dato: **pygame.surface**.
- **interlineado:** Indica la cantidad de píxeles entre cada linea. Tipo de dato: **entero**.
- **espaciado:** Indica la cantidad de píxeles entre cada palabra. Tipo de dato: **entero**.
- **ancho:** indica la distancia desde margen izquierdo hasta texto. Tipo de dato: **entero**.
- **altura:** indica la distancia entre el margen superior hasta la primera linea del texto. Tipo de dato: **entero**.
- **texto:** indica el texto. Tipo de dato: **cadena**.
- size: indica el tamaño de la letra. Tipo de dato: **entero**.

### Funciones:

**__init__():** Esta función se ejecuta al crear un objeto de la clase. Inicializa las variables, y define la posición en que serán ubicadas las palabras, las cuales serán almacenadas como imágenes en la lista **img_palabras**. Como parametros solicita:

- ancho.
- altura.
- texto.
- size.

**Ejemplo:**

```
>>> texto = (64, 340, “Anita lava la tina”, 16)
```

Donde **64** y **340** son el ancho y la altura respectivamente, **“Anita lava la tina”** el texto deseado y **16** el tamaño de la fuente.

**agregar_grupo()**: Método encargado de agregar cada imagen del texto al grupo de sprites correspondiente. Solicita como parámetro un grupo de sprites existente, donde se colocara cada imagen.

**Ejemplo:**

>>> texto = (120, 200, “Capicua”, 12)
>>> texto.agregar_grupo(**grupo_palabras**)

Donde **grupo_palabras** es un grupo de sprites que contiene todos los textos que se mostraran en un momento determinado.

## vocabulario

### Descripción:

Esta clase tiene como finalidad almacenar en una estructura de datos las constantes correspondientes a cada palabra que pueda representar el interprete virtual.

### Estrucura(s) de datos:

Un diccionario con la siguiente estructura:

dic = {“palabra” : (“linea_tiempo”, frame_final)} donde:

- “palabra”: Es la palabra que sera interpretada por el interprete.
- “linea_tiempo”: Es la linea de tiempo correspondiente a la palabra.
- “frame_final”: Es el ultimo fotograma de la linea de tiempo.

**Ejemplo:**

dic = {“nutrientes” : (“action_nutrientes”, 912), “mineral” : (“action_mineral”, 815)}

### Funciones:

**Consultar(palabra):** Esta función devuelve una tupla de valores del diccionario de palabras. Dicha esta formada por una cadena de texto y un entero.

**Ejemplo:**

>>> Consultar (“transporte”)
>>> (“action_transporte”, 780)

## interprete

### Descripción:

Esta clase se encarga de recibir las entradas provenientes del ReDA y de controlar las acciones del interprete virtual.

## Estructura de Datos:

- genero: Es una variable de tipo cadena (string). Puede tomar los siguientes valores: “Hombre” o “Mujer”.
- palabra: Es una variable de tipo cadena (string). Puede tomar los siguientes valores: “fotosíntesis”, “sexual”, “asexual” y “célula”.
- voc: Es un objeto de la clase Vocabulario.
- controlador: Es un objeto derivado del modulo **bge**, mediante el cual de accede a las funciones del modo juego de blender. Para mas detalles vease: http://wiki.blender.org/index.php/Doc:2.6/Manual/Game_Engine.

## Funciones:

**Consultar():** Esta función lee el archivo de configuración **config.ini** y asigna valores a los atributos **genero** y **palabra**. No devuelve ningun valor ni muestra cambios visibles para el usuario.

**Interpretar():** Esta función se encarga de asignar los valores de **genero** y **palabra** al objeto **controlador** para que este controle la representación del interprete virtual.

**Ejemplo:**

>>> interprete.Consultar()

Muestra una pantalla similar: