Imagen: http://edgedfeather.deviantart.com/art/Green-Tree-Python-428853244

Repositorio: https://github.com/lennin92/pythondjango101

Introducción

En esta guía se asume que ya se es conocedor de lenguajes de programación con sintaxis semejante a la de C/C++. Se iniciara con los constructores principales que tienen en común los lenguajes como Java, C/C++ con Python:

- Tipos de datos
- Ciclos
- Lectura de entrada de datos
- Condicionales
- Manejo de excepciones

Luego de conocer cómo hacer un programa simple usando Python, se harán programas con mayor complejidad, cuando se domine la realización de un programa complejo, se harán aplicaciones web usando el framework Django.

Python es un lenguaje de programación interpretado ampliamente usuario de propósitos generales. Es diseñado utilizando una filosofía que se enfatiza en la legibilidad de su código y en una sintaxis que permita expresar conceptos en menos cantidad de líneas de código que en otros lenguajes de programación como Java, C/C++.

Soporta múltiples paradigmas de la programación entre estos POO y estructural, se caracteriza por un sistema de tipado dinámico, un manejo automático de memoria y su largo y compleja biblioteca estándar.

Los intérpretes de Python están disponibles para ser instalados en muchos sistemas operativos, permitiendo la ejecución en una gran cantidad de sistemas. **CPython** es la implementación referencia de Python, que es software libre y tiene un modelo de desarrollo basado en comunidad, así como implementaciones alternativas.

Este curso no busca ser una fuente de referencia para Pythom mas bien; busca dar un sentido de la programación en python usando ejemplos, comparando el lenguaje con alguno de sus competidores como C/C++ o Java.

Semantica y sintaxis.

La sintaxis de Python es un conjunto de reglas que definen como un programa en Python debe ser escrito e interpretado (tanto para humanos y sistemas en tiempo de ejecución). Python fue disenado para ser un lenguaje altamente legible.

Palabras reservadas

and	False	not
as	finally	or
assert	for	pass
break	from	print
class	global	raise
continue	if	return
def	import	True
del	in	try
elif	is	while
else	lambda	with
except	None	yield
exec	nonlocal	

Notas:

- Estas palabras no pueden ser usadas como identificadores o nombres de variables, funciones, clases o paquetes.
- exec y print desde python 3 no son palabras reservadas sino funciones.
- nonlocal es una palabra reservada desde python 3.

Identación

Python utiliza espaciado para delimitar bloques de programa, dividiendolos por nivel de identación. En lugar de puntuaciones o palabras reservadas, utiliza la identacion para indicar la corrida de un bloque.

En lenguajes como C/C++ o Java, los bloques de codigo son distinguidos por un set de llaves “{}”. En varias convenciones para estos lenguajes, los programadores identan el codigo en bloques, para diferenciarlo de el codigo que le rodea.

C/C++	Python

void int potencia(int x, int p){
  if (x==0){
  return 0;
  } else if (p==0){
  return 1;
  } else if (p<0){
  return (1/x)*potencia(x,p+1);
  } else{
  return x*potencia(x,p-1);
  }
}
	def potencia(x, p):
	  if x==0:
	  return 0
	  elif p==0:
	  return 1
	  elif p<0:
	  return (1/x)*potencia(x,p+1)
	  else:
	  return x*potencia(x,p-1)

Flujos de control y sentencias

if	Ejecuta un bloque de codigo condicionalmente, junto a else y elif (una contraccion de else-if)
for	Itera sobre un objeto iterable, capturando cada elemento a una variable local para el uso en el bloque
while	Ejecuta un bloque de codigo siempre que su condicion se evalue en True
try	Permite que se emitan excepciones en el bloque de codigo asociado, para que sean capturadas y manejadas por la clausula except, tambien asegura ejecucion en el bloque de finally
class	Ejecuta un bloque de codigo para ser usado en POO
def	Define una funcion o método
with	Encierra el bloque de codigo a en un asistente de contexto.
pass	Sentencia que permite crear bloques de codigo vacíos.
yield	Usado para retornar el valor de una funcion “generator”

Sentencia if-elif-else

Permite ejecutar bloques de instrucciones siempre que se cumpla una precondición.

Plantilla

```
if condicion:
    # parte True
    pass
elif otra_condicion:
    # parte False para condiciones anteriores
    # pero True para la condicion actual
    pass
else:
    # parte False de todas las condiciones
    pass
```

Ejemplo:

```
from random import randint

VALOR = randint(0,100)

if VALOR%2==0:
    print("el valor %d es multiplo de 2"%(VALOR))
elif VALOR%3==0:
    print("el valor %d es multiplo de 3"%(VALOR))
elif VALOR%5==0:
    print("el valor %d es multiplo de 5"%(VALOR))
else:
    print("el valor %d no es multiplo de 2, 3 ni 5"%(VALOR))
```

Sentencia for

Ciclo que recorre un objeto iterable (lista, tupla, string, set, diccionarios, etc)

Plantilla

```
for elemento in iterable:
    # hacer algo con elemento
    pass
```

Ejemplo

```
CANTIDAD = 10

for VALOR in range(CANTIDAD):
    if VALOR%2==0: print("el valor %d es multiplo de 2"%(VALOR))
    elif VALOR%3==0: print("el valor %d es multiplo de 3"%(VALOR))
    elif VALOR%5==0: print("el valor %d es multiplo de 5"%(VALOR))
    else: print("el valor %d no es multiplo de 2, 3 ni 5"%(VALOR))
```

Sentencia while

Ciclo que se repite mientras se cumpla con una condicion:

Plantilla

```
while condicion:
    # accion a repetir
    pass
```

Ejemplo

```
from random import randint

while True: # este ciclo correra para siempre
    VALOR = randint(0,100)

    if VALOR%2==0:
        print("el valor %d es multiplo de 2"%(VALOR))
    elif VALOR%3==0:
        print("el valor %d es multiplo de 3"%(VALOR))
    elif VALOR%5==0:
        print("el valor %d es multiplo de 5"%(VALOR))
    else:
        print("el valor %d no es multiplo de 2, 3 ni 5"%(VALOR))
        break # permite forzar cierre del ciclo
```

Sentencia try

Permite manipular el codigo que en algunas condiciones puede producir error

Plantilla

```
try:
    # hacer algo que pueda generar una excepcion
    pass
except TipoExcepcion as e:
    # hacer algo con la excepcion e
    pass
```

Ejemplo

```
try:
    a = 2562/0
    print(a)
except Exception as e:
    print("ERROR!")
    print(e)
```

Sentencia def

Permite crear funciones o metodos de clase.

Plantilla

```
def nombre_funcion(parametros):
    # hacer algo con los parametros y retornar algo (si es deseable)
    pass
```

Ejemplo

```
def es_multiplo(valor, multiplicidad):
    return valor%multiplicidad==0

def es_mult_2(valor):
    return es_multiplo(valor, 2)

def es_mult_3(valor):
    return es_multiplo(valor, 3)

def es_mult_5(valor):
    return es_multiplo(valor, 5)

from random import randint

VALOR = randint(0,100)

if es_mult_2(VALOR):
    print("el valor %d es multiplo de 2"%(VALOR))
elif es_mult_3(VALOR):
    print("el valor %d es multiplo de 3"%(VALOR))
elif es_mult_5(VALOR):
    print("el valor %d es multiplo de 5"%(VALOR))
else:
    print("el valor %d no es multiplo de 2, 3 ni 5"%(VALOR))
```

Sentencia class

Permite declarar una clase.

Plantilla

```
class NombreClase:
    atributo1 = 1
    atributo2 = 1

    def __init__(self, parametros):
        # constructor
        pass

    def metodo1(self, parametros):
        #hacer algo
        pass
```

Ejemplo

```
class Libro:
    titulo = None
    autor = None
    anio = None
    genero = None
    edicion = None
    editorial = None
    sinopsis = None

    def __init__(self, titulo, autor, editorial):
        self.titulo = titulo
        self.autor = autor
        self.editorial = editorial

    def __str__(self):
        s = "<Libro  titulo: %s, autor:%s, editorial %s>"
        return s % (self.titulo, self.autor, self.editorial)


l = Libro('Cien años de soledad', 'Gabriel García Márquez', 'Alfaguara')
print(l)
```

Guías de trabajo

Guia 1 Introduccion a python

Objetivo General: Que se obtenga un dominio general del uso de python.

Objetivos Especificos:

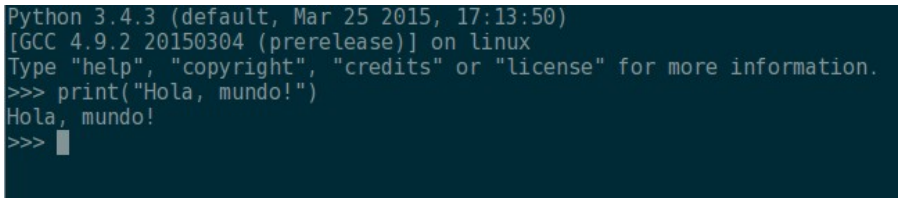
1. Usar el interprete interactivo de python.
2. Realizar programas con ciclos y desiciones.
3. Realizar y ejecutar scripts de python.
4. Realizar operaciones con cadenas.

En esta giua se provee de suficiente informacion para iniciar a escribir programas en Python. La primera recomendación es que se instale Python (si es que aun no se ha instalado), se puede usar cualquier editor de texto para escribir codigo en python.

Para abrir el intérprete de python, se debe dar a la consola (o cmd en windows), la siguiente orden:

```
python
```

1- Ejemplo, imprimir en la pantalla “Hola, mundo!”



2- Ejemplo, usar python como calculadora

```
Python 3.4.3 (default, Mar 25 2015, 17:13:50)
[GCC 4.9.2 20150304 (prerelease)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 3+3
6
>>> 9+6
15
>>> 9-6
3
>>> 9+-9
0
>>> 9//4
2
>>> 9/4
2.25
>>> 2**3
8
>>> 2+9-8+9*6
57
>>>
```

3- Ejemplo, crear un script que permita obtener los primeros 10 numeros primarios y los imprima.

```
Python 3.4.3 (default, Mar 25 2015, 17:13:50)
[GCC 4.9.2 20150304 (prerelease)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> lim = 10
>>> primos = [1,2,3]
>>> n = 4
>>> while len(primos)<=lim:
...     cnt=0
...     for p in primos:
...         if n%p==0: cnt+=1
...     if cnt==2: primos.append(n)
...     n+=1
...
>>> print("Los primeros %d numeros primos son:"%(lim))
Los primeros 10 numeros primos son:
>>> print(primos)
[1, 2, 3, 4, 9, 10, 14, 15, 21, 22, 26]
>>>
```

Al parecer en el codigo hay un error lógico, puesto que esta imprimiendo numeros que claramente no son primos, por ejemplo 1, 4, 9, ….

Discutir cual es el posible error lógico. Pista: la solucion se basa en que un numero primo debe ser divisible entre el mismo y 1.

4- Ejemplo, crear una funcion que cuente el numero de “a” que tiene una cadena.

```
Python 3.4.3 (default, Mar 25 2015, 17:13:50)
[GCC 4.9.2 20150304 (prerelease)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> def  contar_as(texto):
...     cnt = 0
...     for char in texto:
...         if char=='a':
...             cnt += 1
...     return cnt
...
>>> cadena = "La Ardilla tiene neumonia"
>>> contar_as(cadena)
3
>>>
```

La funcion está bien escrita, sin embargo no da el resultado correcto, debería de imprimir 4. Discutir cual podria ser el error.

5- Ejemplo, crear un script llamado “funcion.py” y definir en este una funcion llamada “contar_letra” cuyos parametros sean una cadena de texto y un carácter, y retorne la cantidad de veces que el carácter se repita en la cadena

```
def contar_letra(cadena, letra):
        cnt = 0
        for char in cadena:
                if char==letra: cnt += 1
        return cnt
```

Para poderse usar solo debe importarse el archivo y llamar a la funcion:

```
Python 3.4.3 (default, Mar 25 2015, 17:13:50)
[GCC 4.9.2 20150304 (prerelease)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import funcion
>>> TEXTO = """
... Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque d
elit id metus condimentum pharetra. Quisque luctus elit vitae dui elem
idunt sit amet feugiat felis. Mauris tempus massa nec quam suscipit bl
am ultricies, ante ut iaculis venenatis, erat sapien viverra augue, eg
rper purus libero et ligula. Vivamus fringilla tempor erat quis tristi
 mattis, tellus eget ullamcorper molestie, mauris enim fermentum enim,
sis neque lectus eget purus. Morbi sed blandit nunc. Proin quis risus
 eleifend luctus """
>>> LETRA = "u"
>>> funcion.contar_letra(TEXTO, LETRA)
47
>>>
```

Ejercicio 1. En el archivo “funcion.py” cambiar la funcion “contar_letra” para que funcione tanto “Case sensitive” como no “Case Sensitive”, para ello agregar un parametro opcional “sensitivo” con valor por defecto = False; si este parametro es True, debera considerar las mayusculas diferentes de las minusculas, si el valor del parametro es False, debera tomar las mayusculas y minusculas por igual.

Ejercicio 2. Investigar sobre los diccionarios en python, como se usan y para que pueden servir.

Ejercicio 3. En el archivo “funcion.py” crear una funcion llamada “obtener_cuentas” con un solo parametro llamado “texto”, que retorne un diccionario, cuyas llaves sean cada palabra dentro de la variable “texto” y su valor sea el numero de veces que se repite esa palabra dentro de la variable.

Ejercicio 4. Investigar sobre el uso de archivos en python, ¿Como se leen? y ¿Como se escriben?.

Ejercicio 5. En el archivo “funcion.py” crear una funcion llamada “obtener_cuentas_archivo”, cuyo parametro sea la direccion de un archivo plano; esta funcion debera obtener el mismo resultado de la funcion “obtener_cuentas” del contenido del archivo parametro

Guia 2 Programacion orientada a objetos

Objetivo General: Desarrollar un sistema para gestionar liricas de canciones.

Objetivos Específicos:

1. Aprender a programar clases en python.
2. Implementar subsistema propio de persistencia.
3. Aprender a realizar sistemas usando el patron MVC.

En esta guia se desarrollará un sistema que gestione liricas de canciones, en una version de escritorio.

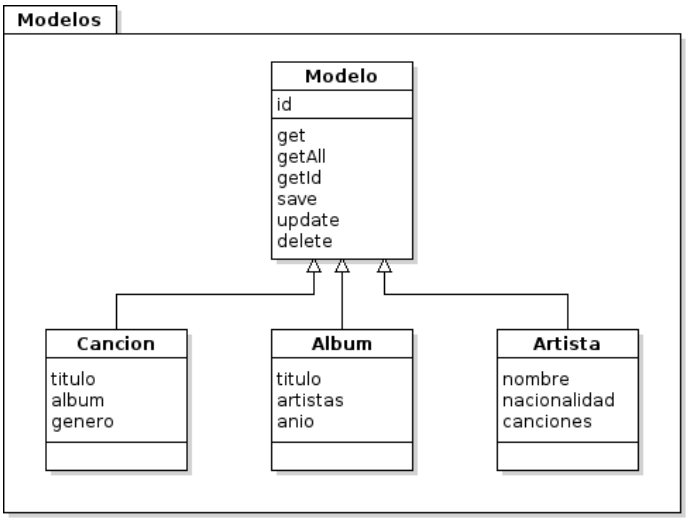
Para realizar esta guía es necesario tener conocimientos previos sobre:

- UML
- Programar clases básicas en python
- Diccionarios en Python
- Listas en Python
- Uso de archivos en python (lectura y escritura)

Ejercicios

Ejercicio 1. Investigar como crear un paquete en python. Crear un directorio con el nombre “sisLyric” este sera el directorio del proyecto. Dentro de este directorio crear 3 paquetes “Modelos”, “Controladores” y “Vistas”

Ejercicio 2. Programar las clases del siguiente diagrama. Para los metodos de la clase Modelo dejar sin implementar con la palabra reservada “pass”. Investigar como implementar herencia en python.



Ejercicio 3. Investigar sobre el uso del module pickle en python. ¿Para qué sirve? ¿Cómo se usa?

Ejercicio 4. En el Paquete “Modelos”, dentro del archivo Modelo.py crear 1 diccionario (vacio) llamado “MODELOS” y 2 funciones “cargarModelos” y “persistirModelos”.

- La función persistirModelos debera usar el modulo pickle de python para persistir el diccionario MODELOS en un archivo de nombre “bd”.
- La función cargarModelos deberá usar el modulo pickle de python para cargar el objeto de el archivo de nombre “bd” a la variable MODELOS.

Para el diccionario MODELOS, las llaves seran enteros que se iran incrementando a medida se llene el diccionario con objetos a persistir.

Ejercicio 5. Dentro de la clase modelo implementar las funciones get, getAll, getId, save, update y delete, de la siguiente forma:

- get: (metodo de clase) debera usar un parametro llamado “sid”, debera retornar el elemento cuyo id sea el valor de “sid”.
- getAll: (metodo de clase) debera retornar una lista con todos los objetos del modelo que la llame.
- save: debera persistir el objeto que lo llame.
- update: debera persistir las actualizaciones hechas al objeto que lo llame.
- delete: debera eliminar el objeto que lo llame.
- getId: debera obtener el id del objeto que lo llame.

Ejercicio 6. Impementar los controladores de album, cancion y artista; usar el siguiente ejemplo como plantilla (si tienes acceso, puedes usar el que esta en el repositorio de github).

```python
# -*- coding: utf-8 -*-
"""
@author: lennin

Controladores Album

"""
from Modelos import Album

def get_all_albums():
    return Album.getAll()

def save_album(album):
    if isinstance(album, Album):
        album.nombre = album.nombre.lower()
        album.nacionalidad = album.nacionalidad.lower()
        album.save()

def get_album_by_id(ida):
    return Album.get(ida)

def get_album_with_similarity(sim):
    albums = []
    simi = sim.lower()
    for a in Album.getAll():
        if simi in a.titulo:
            albums.append(a)
    return albums

def get_album_by_nombre(nombre):
    albums = []
    simi = nombre.lower()
    for a in Album.getAll():
        if simi in a.nombre:
            albums.append(a)
    return albums
```

Ejercicio 7. Crear las Vistas, usando la siguiente como ejemplo (puedes encontrarla en el repositorio de GitHub)

```python
from Controladores import AlbumControlador

from Vistas.BaseVista import VistaBase, entrada, salir

from Modelos import Album

def consultar_todos():
    print("ALBUMS: ")
    print(AlbumControlador.get_all_albums())

def consultar_por_id():
    aid = entrada("ID A BUSCAR: ")
    a = AlbumControlador.get_album_by_id(aid)
    print("ALBUM: ")
    print(a)

def consultar_por_similares():
    sim = entrada("FILTRO: ")
    a = AlbumControlador.get_album_with_similarity(sim)
    print("ALBUMS: ")
    print(a)

def agregar_album():
    a = Album()
    a.titulo = entrada("TITULO: ")
    a.artistas = entrada("ARTISTAS: (id separados por espacio)").split(" ")
    a.anio = entrada("ANIO: ")
    AlbumControlador.save_album(a)
    print("ARTISTA AGREGADO (id %d)"%(a.getId()))

class VistaAlbum(VistaBase):
    menu = {
        1: ("CONSULTAR TODOS LOS ALBUMS", consultar_todos),
        2: ("BUSCAR ALBUM POR ID", consultar_por_id),
        3: ("BUSCAR POR SIMILITUD", consultar_por_similares),
        4: ("AGREGAR", agregar_album),
        5: ("salir", salir)
    }
```

Ejercicio 8. Ejecutar la aplicación

```
[lennin@LenninPC guia2]$ python main.py
[1] GESTION ARTISTAS
[2] GESTION ALBUMS
[3] GESTION DE CANCIONES
[4] SALIR
Seleccione opcion: 2
[1] CONSULTAR TODOS LOS ALBUMS
[2] BUSCAR ALBUM POR ID
[3] BUSCAR POR SIMILITUD
[4] AGREGAR
[5] salir
Seleccione opcion: 1
ALBUMS:
[<class Album id: 2>, <class Album id: 5>, <class Album id: 8>]
[1] CONSULTAR TODOS LOS ALBUMS
[2] BUSCAR ALBUM POR ID
[3] BUSCAR POR SIMILITUD
[4] AGREGAR
[5] salir
Seleccione opcion:
```

Guia 3 Django

Objetivo General: Desarrollar un juevo de memoria utilizando el framework django.

Objetivos Específicos:

1. Aprender a crear proyectos usando Django.
2. Aprender a crear aplicaciones usando Django.

En esta guia se desarrollara un juego de memoria usando el framework django, para ello se creara un perfil de usuario y se guardaran los puntajes.

Para realizar esta guía es necesario tener conocimientos previos sobre:

- Programacion basica-media en python
- POO en python
- Diccionarios de python

Ejemplo 1. Creacion de proyectos en Django.
Investigar como instalar Django y/o si esta instalado.

```
[lennin@LenninPC ~]$ django-admin startproject guia_3
[lennin@LenninPC ~]$
```

Si el comando se ejecuto correctamente, se debió haber creado la carpeta llamada “guia_3”.

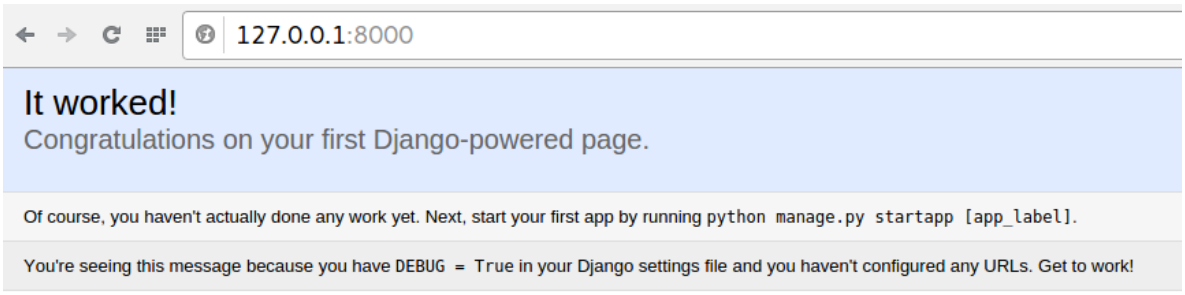
Ejemplo 2. Creacion de una applicacion en un proyecto de django.

1. Cambiar al directorio del proyecto creado.
2. Crear la aplicación.

```
[lennin@LenninPC ~]$ cd guia_3
[lennin@LenninPC guia_3]$ python manage.py startapp memoria
[lennin@LenninPC guia_3]$
```

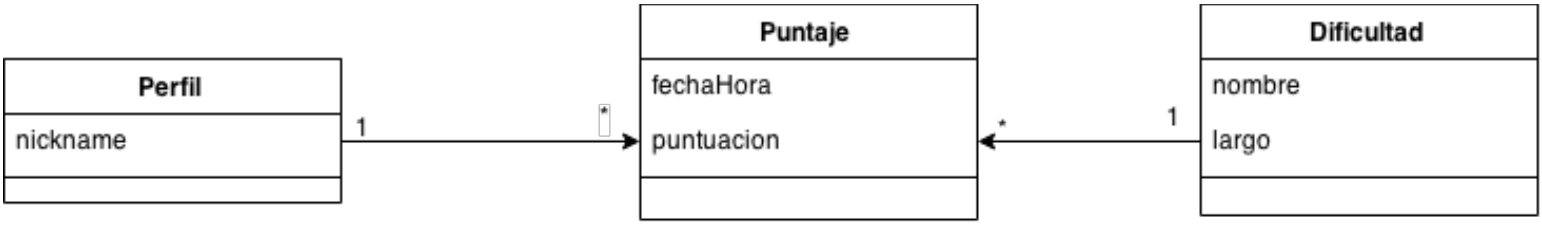
Ejemplo 3. Ejecutar el servidor de prueba y mostrar en el navegador.

```
[lennin@LenninPC guia_3]$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are app
lied.
Run 'python manage.py migrate' to apply them.

April 27, 2015 - 05:21:26
Django version 1.8, using settings 'guia_3.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```



Ejercicio 1. Investigar en que parte de la aplicación se deben crear los modelos.

Ejercicio 2. Desarrollar el siguiente modelo.



Usar como plantilla:

```
from django.db import models


class Perfil(models.Model):
    nickname = models.CharField(max_length=25)


class Puntaje(models.Model):
    pass

class Dificultad(models.Model):
    pass
```

Investigar los tipos de datos que se usan para persistir modelos en Django
https://docs.djangoproject.com/en/1.8/ref/models/fields/#field-types