



Clase 25. Python

***¿Y ahora qué?***

***RECUERDA PONER A GRABAR LA  
CLASE***





## ***OBJETIVOS DE LA CLASE***

- Conocer las limitaciones de este curso para entender por donde deberán continuar para seguir avanzando en Python.
- Aprender a subir sus proyectos a un servidor.
- Articular con otros temas fundamentales como Desarrollo Web y bases de datos.

# ***CRONOGRAMA DEL CURSO***

## Clase 24



### **Playground Avanzado (parte III)**



DOCUMENTOS CASOS DE  
PRUEBAS



AGREGAR UNA IMÁGEN



EDICIÓN DE USUARIO



UNI TEST

## Clase 25



### **¿Ahora qué?**



PLAYGROUND AVANZADO



ENTREGA FINAL



# ***PLAYGROUND AVANZADO***

Compartir entre los estudiantes los desafíos de clases 22, 23 y 24.

**TIEMPO: 20 MIN**



# ***ACUERDOS***



**Presencia**



**Escucha Activa**



**Apertura al  
aprendizaje**



**Todas las voces**

# ***PLAYGROUND AVANZADO***

Actividades  
colaborativas



**Consigna:** Elegir un representante por cada grupo que deberá compartir con el resto de la clase el proceso de resolución de los desafíos genéricos de clases 22, 23 y 24; correspondientes a la unidad de PlayGround Avanzado.

**NOTA:** usaremos los breakouts rooms. El tutor/a tendrá el rol de facilitador/a.

***CODER HOUSE***

# ***HEROKU - PYTHONANYWHERE***



***HEROKU***

# ***Heroku***

Heroku es una plataforma que ayuda a los desarrolladores a ejecutar y mantener sus proyectos sin preocuparse por la gestión de la infraestructura.

Esto incluye 👉 bases de datos, seguridad, networking, logging y monitoreo.

Con Heroku podrás tener online tu proyecto en pocos minutos.



***CODER HOUSE***

# ***Heroku***

La popularidad de Heroku ha crecido en los últimos años, principalmente por su facilidad de uso.

Fue creada por desarrolladores para desarrolladores, lo que la convierte en la mejor opción para muchos proyectos de desarrollo.



***CODER HOUSE***



# ***Heroku - Crear cuenta***



Siempre lo primero que deberán hacer es tener una cuenta, pueden hacerlo fácilmente desde [Heroku](#)



Tendrán disponible toda la documentación oficial de Heroku con Django desde [Heroku - Django](#)



***CODER HOUSE***



# ***Heroku - Librerías***

- 1) Para subir tu proyecto deberás instalar algunos complementos en tu entorno, por medio de `pip`.

```
pip install dj-database-url gunicorn whitenoise
```

```
pip freeze > requirements.txt
```

- 👉 Luego deberás abrir el archivo `requirements.txt` y agregar al final

```
psycopg2
```



# ***Heroku - Nuestra App***

2) Heroku necesita saber qué tipo de app es la nuestra y cómo ejecutarla. Para eso creamos el archivo **Procfile** (sí, sin extensión y con mayúscula) dentro de nuestro proyecto y agregamos al archivo la siguiente línea: **web: gunicorn miSitio.wsgi**

Luego instalar por medio de pip y actualizar los requisitos 🙌

```
pip install gunicorn
```

```
pip freeze > requirements.txt
```

***CODER HOUSE***



# ***Heroku - Python***

- 3) Deberás avisarle a Heroku con cuál Python trabajaste. Vamos a crear otro archivo dentro de nuestro proyecto llamado `runtime.txt` y le agregamos la siguiente línea:

```
python-3.5.2
```



# ***Heroku - Settings.py***

- 4) Hay que cambiar un par de cosas para que nuestro proyecto funcione correctamente en Heroku.

Primero agregaremos estas líneas hasta el final de nuestro **archivo settings.py**:

```
import dj_database_url
db_from_env = dj_database_url.config(conn_max_age=500)
DATABASES['default'].update(db_from_env)

STATICFILES_DIRS = ( os.path.join(BASE_DIR, 'static'), )

STATICFILES_STORAGE = 'whitenoise.django.GzipManifestStaticFilesStorage'
```





# ***Heroku - wsgi.py***

5) Agregar todo lo que falte en este archivo para que se vea así:

```
import os
from django.core.wsgi import get_wsgi_application
os.environ.setdefault("DJANGO_SETTINGS_MODULE",
                    "miSitio.settings")
application = get_wsgi_application()
```



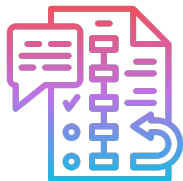
# ***Heroku - Heroku CLI***

6) Luego empezamos a subir a Heroku, para eso necesitamos instalar [Heroku CLI](#)

```
heroku login
```

```
heroku create
```

Paso a paso 🙌



```
git add -A
```

```
git commit -m "listos"
```

```
git push heroku master
```

```
heroku open
```

***CODER HOUSE***



# Deploy Django App on Heroku



***PYTHONANYWHERE***

# ***Pythonanywhere***

Hosting dedicado únicamente a aplicaciones basado en Python.

Contiene un entorno muy amigable, acceso bash mediante la web, administración de base de datos, entorno virtual; entre otras muchas ventajas. Para aplicaciones de poco tráfico es gratuito 😊



Página oficial: [Pythonanywhere](https://pythonanywhere.com)

Documentación oficial: [Documentación](https://pythonanywhere.com/documentation)



# Deploying Django Web Application to PythonAnywhere.com by Srikanth Pragada



***CODER HOUSE***

[How to deploy Django Application to PythonAnywhere.com](#)

# ***PYTHONANYWHERE VS HEROKU***

# ***Pythonanywhere vs heroku***

<b>Ventajas Heroku</b>	<b>Ventajas Pythonanywhere</b>
<ul style="list-style-type: none"><li>● Despliegue sencillo.</li><li>● Gratis para proyectos paralelos.</li><li>● Enorme ahorro de tiempo.</li><li>● Escalado simple.</li><li>● Se requieren habilidades bajas en DevOps.</li><li>● Configuración fácil.</li><li>● Complementos para casi todo.</li><li>● Apto para principiantes.</li><li>● Mejor para startups.</li></ul>	<ul style="list-style-type: none"><li>● Aplicaciones web.</li><li>● Configuración fácil.</li><li>● Superfácil de usar.</li><li>● Acceso a Shell.</li><li>● Gran apoyo.</li><li>● Plan gratuito.</li><li>● Muchas cosas como Python están preinstaladas.</li></ul>



# ***Pythonanywhere vs heroku***

<b>Ventajas Heroku</b>	<b>Ventajas Pythonanywhere</b>
<ul style="list-style-type: none"><li>● Curva de aprendizaje baja.</li><li>● Alojamiento Postgres.</li><li>● Fácil de agregar colaboradores.</li><li>● Desarrollo más rápido.</li><li>● Impresionante documentación.</li><li>● Céntrese en el producto, no en la implementación.</li><li>● Retroceso simple.</li><li>● Fácil integración.</li></ul>	

# ***Pythonanywhere vs heroku***

<b>Desventajas Heroku</b>	<b>Desventajas Pythonanywhere</b>
<ul style="list-style-type: none"><li>● No mucha flexibilidad.</li><li>● No hay opción de MySQL utilizable.</li><li>● Almacenamiento.</li><li>● Bajo rendimiento en el nivel gratuito.</li><li>● El soporte 24/7 cuesta \$ 1,000 por mes</li></ul>	<ul style="list-style-type: none"><li>● Sin acceso de root.</li><li>● Comunidad realmente pequeña.</li></ul>



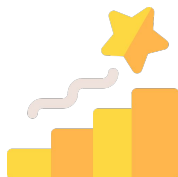
# ***EL SECRETO DEL PROGRAMADOR***

Para cosas sencillas PythonAnywhere, para proyectos complejos y cooperativos Heroku, incluso para poner en producción.

# ***OTRAS ALTERNATIVAS***

# ***Otras alternativas***

Hay muchos otros servicios web semejantes 😊



Aquí los diez más populares: [Deploy Web Django](#)



De esa lista la alternativa más completa es la [nube de Amazon](#)

***CODER HOUSE***

# ***PENDIENTES DE PYTHON***

# ***Generadores***

Los generadores son una forma sencilla y potente de iterador.

Un generador es una función especial que produce secuencias completas de resultados en lugar de ofrecer un único valor.



En apariencia es como una función típica pero en lugar de devolver los valores con `return` lo hace con la declaración `yield`.

Hay que precisar que el término generador define tanto a la propia función como al resultado que produce.



Una característica importante es que tanto las variables locales como el punto de inicio de la ejecución se guardan automáticamente entre las llamadas sucesivas que se hagan al generador.

Es decir, a diferencia de una función común, una nueva llamada a un generador no inicia la ejecución al principio de la función, sino que la reanuda inmediatamente después del punto donde se encuentre la última declaración `yield` (que es donde terminó la función en la última llamada).





# *Generadores*

Varios ejemplos para empezar: [Generadores](#)

```
def funcion_generadora_prints():  
    print("GENERADOR: Se va a generar un PRIMER dato")  
    yield "valorGenerado1"  
  
    print("GENERADOR: Se va a generar un SEGUNDO dato")  
    yield "valorGenerado2"  
  
    print("GENERADOR: Se va a generar un TERCER dato")  
    yield "valorGenerado3"  
  
    print("GENERADOR: Termina y lanzará automáticamente la excepción StopIteration")
```



# ***Lambda***

En Python, una función Lambda se refiere a una **pequeña función anónima**. Las llamamos “funciones anónimas” porque técnicamente carecen de nombre.

Al contrario que una función normal, no la definimos con la palabra clave estándar `def` que utilizamos en Python. En su lugar, **las funciones Lambda se definen como una línea que ejecuta una sola expresión.**



```
#Creo una función Lambda para pasar los grados Celsius a Kelvin.  
to_kelvin = lambda x: (x + 273,15)  
  
#Aplico la Lambda a la columna [Kelvin] con el método apply()  
df = df['Kelvin'].apply(to_kelvin)  
|
```

***CODER HOUSE***



# ***Filter()***

Como su nombre indica filter significa filtrar, ya que a partir de una lista o iterador y una función condicional, es capaz de devolver una nueva colección con los elementos filtrados que cumplan la condición.

```
def multiple(numero):    # Primero declaramos una función condicional
    if numero % 5 == 0:  # Comprobamos si un numero es múltiple de cinco
        return True     # Sólo devolvemos True si lo es

numeros = [2, 5, 10, 23, 50, 33]

filter(multiple, numeros)
```

➡ Supongamos que tenemos una lista varios números y queremos filtrar, quedándonos únicamente con los múltiplos de 5.



# Maps

Esta función trabaja de una forma muy similar a `filter()`, con la diferencia que en lugar de aplicar una condición a un elemento de una lista o secuencia, **aplica una función sobre todos los elementos y como resultado se devuelve un iterable de tipo map:**



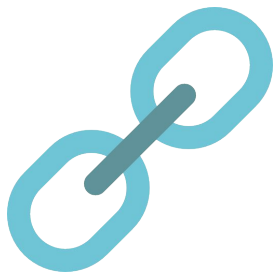
```
def doblar(numero):  
    return numero*2  
  
numeros = [2, 5, 10, 23, 50, 33]  
  
map(doblar, numeros)
```

**CODER HOUSE**



# ***Expresiones regulares***

Cuando manejamos texto en Python, una de las operaciones más comunes es la búsqueda de una subcadena; ya sea para obtener su posición en el texto o simplemente para comprobar si está presente.



Si la cadena que buscamos es fija, los métodos como `find()`, `index()` o similares nos ayudarán. Pero si buscamos una subcadena con cierta forma, este proceso se vuelve más complejo.



# ***Expresiones regulares***

Al buscar direcciones de correo electrónico, números de teléfono, validar campos de entrada, o una letra mayúscula seguida de dos minúsculas y de 5 dígitos entre 1 y 3; es necesario recurrir a las Expresiones Regulares, también conocidas como Patrones.

**.\*** : cualquier cadena, de cualquier largo (incluyendo una cadena vacía)

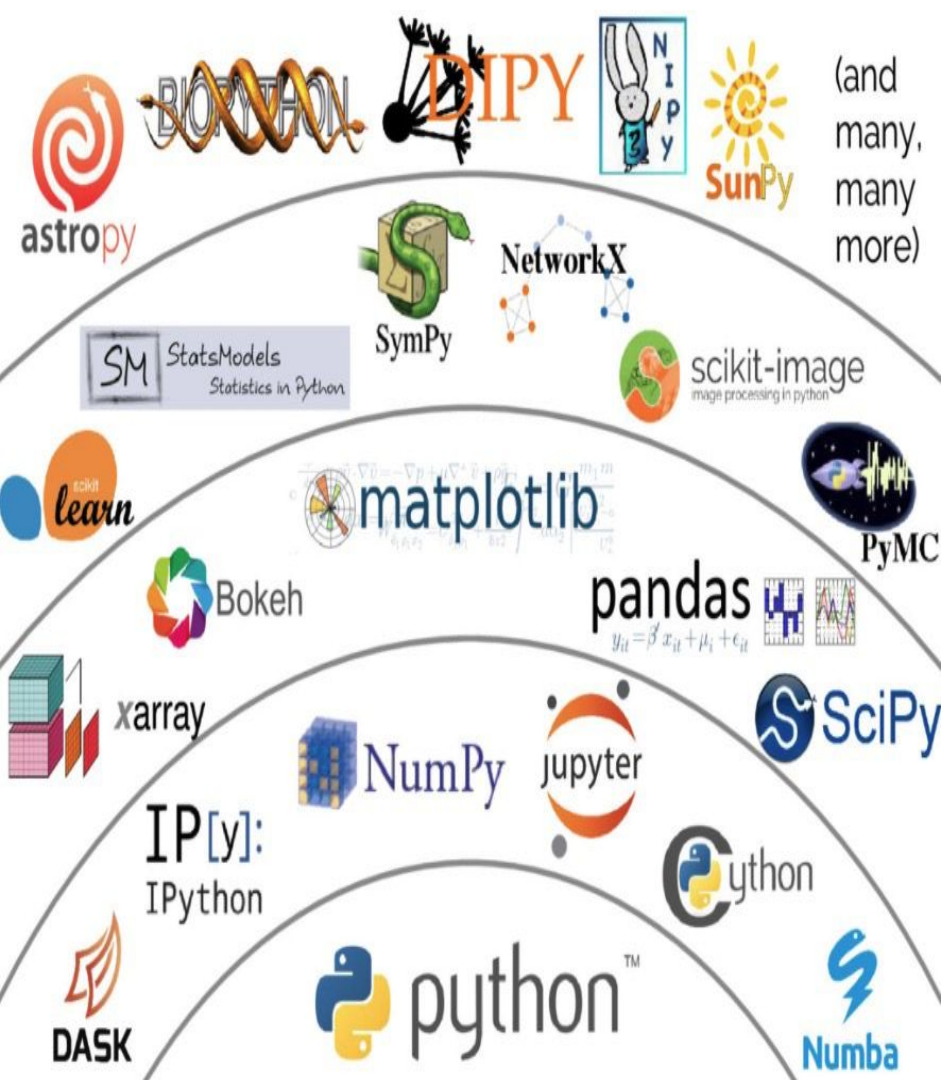
**[a-z]{3,6}**: entre 3 y 6 letras minúsculas

**\d{4,}**: al menos 4 dígitos

**.\*hola!?:** una cadena cualquiera, seguida de hola, y terminando (o no) con un !



# ***EXPLOTACIÓN DE DATOS***



# ***Explotación de datos***

Python tiene un gran poder de cómputo para manejar grandes volúmenes de datos, tiene varias librerías asociadas, recomendamos empezar por Pandas.

***CODER HOUSE***



***MEJORAR LA ESTÉTICA***

# ***Mejorar la estética***

Hemos realizado un proyecto web muy funcional y útil, pero jamás nos fijamos en lo mal que se veía. Para mejorar todo su aspecto es fundamental conocer más sobre HTML, incorporar CSS y en menor medida JS y SASS.

En el siguiente curso de CoderHouse pueden ver todos esos temas: [Desarrollo WEB](#)



***CODER HOUSE***

# ***OPTIMIZAR BASE DE DATOS***

# ***Optimizar Base de datos***

Si quieren seguir trabajando con Python y Django verán que SQLite es ideal para aprender, pero su performance y estabilidad es muy baja, se recomienda pasar a PostgreSQL.



# ***Optimizar Base de datos***



PostgreSQL es un sistema de base de datos relacional de alta disponibilidad. Es capaz de funcionar de manera estable en el servidor y, por lo tanto, resulta robusto, una de las principales características que buscan las empresas. Además, es consistente y tolerante a fallos. Es compatible con el modelo relacional, ya que asegura siempre su integridad referencial.

# ***Optimizar Base de datos***

De forma más general se puede estudiar **mySQL** que es ideal para proyectos pequeñas y adaptable a cualquier lenguaje Backend.



Por ejemplo en este curso de CoderHouse: [SQL](#)



***CODER HOUSE***



# ***RECOMENDACIONES FINALES***

***CODER HOUSE***



# ***Recomendaciones Finales***

- Mantente al tanto de las actualizaciones de todas las tecnologías.
- No te quedes con lo aprendido aquí, el mundo es muy grande.
- Practica, practica y practica.
- Establece alcances con cada proyecto.
- Planifica
- Trabaja organizadamente
- Da tiempos estimados con sinceridad





# ***ENTREGA DEL PROYECTO FINAL***

Deberás entregar tu proyecto final.



# ***ENTREGA FINAL***

## ***Crear web similar a un blog.***

- Se deberá realizar en duplas o tríos, crearás una aplicación web estilo blog programada en Python en Django. Esta web tendrá admin, perfiles, registró, páginas y formularios.
- La entrega se realizará enviando el **link a GitHub**, en el **readme** de Github deberá estar el **nombre completo de los tres/dos participantes** y una descripción de dos o tres renglones contando **qué hizo cada uno**.
- En el github debe haber un **video o link a vídeo** donde nos muestran su web funcionando en **no más de diez minutos**.



# ***ENTREGA FINAL***

## ***Crear web similar a un blog.***

- Dentro del Github deberá existir una carpeta con por lo menos 3 casos de pruebas debidamente documentados.
- Contar con algún acceso visible a la vista de "Acerca de mí" donde se contará acerca de los dueños de la página manejado en el **route about/**. (En castellano un “acerca de mí” que hable un poco de los creadores de la web y del proyecto).



# ***ENTREGA FINAL***

## ***Crear web similar a un blog.***

- Contar con algún acceso visible a la vista de blogs que debe alojarse en el **route pages/**.  
(Es decir un html que permite listar todos los blogs de la BD, con una información mínima de dicho blog).
- Acceder a una pantalla que contendrá las páginas. Al clicar en “Leer más” debe navegar al detalle de la page mediante un **route pages/<pageld>**. (O sea al hacer click se ve más detalle de lo que se veía en el apartado anterior).



# ***ENTREGA FINAL***

## ***Crear web similar a un blog.***

- Si no existe ninguna página mostrar un "No hay páginas aún". (Aclarando, si en la página hacemos clic en algún lugar que no existe que diga eso, o que lleve a un html con esos mensaje, no dejar botones que no responden).
- Para crear, editar o borrar las fotos debes estar registrado como Administrador.
- Cada blog, es decir cada model Blog debe tener como mínimo, un título, subtítulo, cuerpo, autor, fecha y una imagen (mínimo y obligatorio, puede tener más).



# ***ENTREGA FINAL***

## ***Crear web similar a un blog.***

- Piezas sugeridas, no hace falta que estén todas, pero tiene que haber por lo menos un CRUD completo y el módulo de Login debe ser sólido:

- NavBar
- Home
- About
- Pages
- Login
- Signup

- Messages
- Profile
- Logout
- Get pages
- Get page

- Create page
- Update Page
- Delete page
- Get profile
- Update profile



# ***ENTREGA FINAL***

## ***Requisitos bases***

- Inicio: Al momento de ingresar a la app en la ruta base **'/'**.
- Visualizar el **home** del blog.
- Poder listar todas las páginas del blog, poder ver en detalle cada una, poder crear, editar o borrar páginas del blog.
- Las páginas están formadas por un título, un contenido en editor de texto avanzado (ckeditor por ejemplo), una imagen, fecha de posteo de la imagen.



# ***ENTREGA FINAL***

## ***Requisitos bases***

- Tener una app de registro donde se puedan registrar usuarios en el route accounts/signup, un usuario está compuesto por: email - contraseña - nombre de usuario.
- Tener una app de login en el route accounts/login/ la cual permite loguearse con los datos de administrador o de usuario normal.
- Tener una app de perfiles en el route accounts/profile/ la cual muestra la info de nuestro usuario y permite poder modificar y/o borrar: imagen - nombre - descripción - un link a una página web - email y contraseña.





# ***ENTREGA FINAL***

## ***Requisitos bases***

- Contar con un admin en route **admin/** donde se puedan manejar las apps y los datos en las apps.
- Tener una app de mensajería en el route **messages/** para que los perfiles se puedan contactar entre sí.

**NOTA: No hace falta que sean APPs separadas, con dos APP estarán bien.**



# ***ENTREGA FINAL***

## ***Recomendaciones extras***

Los requisitos extra *pro-coders* no se incluyen en los criterios de evaluación.

Los requisitos extra son funcionalidades opcionales que no se incluyen en los criterios de evaluación, pero si te falta diversión y quieres agregar valor a tu proyecto... ¡bajo la única **condición** de que **lo que incluyas debe funcionar!**

- Messenger y like - integración otra db
- Subida a un servidor

No es necesario ni recomendado.

- Utilizar Python puro para el proyecto final (se espera el uso de Django).



# ***ENTREGA FINAL***

## ***Pregunta frecuente***

En caso que no quieran hacer una Web simil Blog, pueden, pero deberá tener la misma estructura el modelo básico, título, subtítulo, texto, imagen/es, autor, fecha. Y la web debe tener un funcionamiento similar.

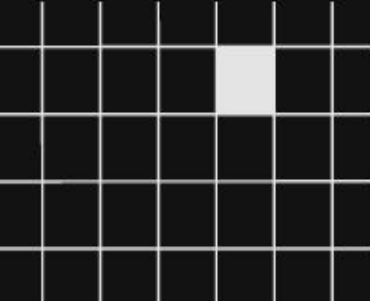
***¿PREGUNTAS?***





# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Heroku
  - PythonAnyWhere
  - Pendientes de Python
- 



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDO LA EDUCACIÓN***

***CODER HOUSE***

***CODER HOUSE***

***¡GRACIAS POR ESTUDIAR CON  
NOSOTROS!***

---