Clase 25. ¿Ahora qué?

**Heroku - Pythonanywhere**

**Heroku**

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

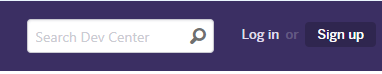
Esto incluye 👉 bases de datos, seguridad, networking, logging y monitoreo. Con Heroku podrás tener online tu proyecto en pocos minutos.

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.

Siempre lo primero que deberán hacer es tener una cuenta, pueden hacerlo fácilmente desde [Heroku](https://www.heroku.com/)

Tendrán disponible toda la documentación oficial de Heroku con Django desde [Heroku - Django](https://devcenter.heroku.com/categories/working-with-django)



**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==2.6.2**

**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**

**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()**

**from whitenoise.django import DjangoWhiteNoise**

**application = DjangoWhiteNoise(application)**

**Heroku - Heroku CLI**

**6)** Luego empezamos a subir a Heroku, para eso necesitamos instalar [Heroku CLI](https://devcenter.heroku.com/articles/heroku-cli)

Paso a paso 👉

heroku create

heroku login

git add -A

git commit -m "listos"

git push heroku master

heroku open

[Para conocer más, puedes hacer click en este link](https://www.youtube.com/watch?v=_3AKAdHUY1M)

**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://www.pythonanywhere.com/)

Documentación oficial: [Documentación](https://help.pythonanywhere.com/pages/DeployExistingDjangoProject/)

[Para conocer más puedes hacer click en este link](https://www.youtube.com/watch?v=H7GTDwgXDBc&t=1s)

**Pythonanywhere vs Heroku**

| **Ventajas Heroku** | **Ventajas Pythonanywhere** |
| --- | --- |
| * Despliegue sencillo. * Gratis para proyectos paralelos. * Enorme ahorro de tiempo. * Escalado simple. * Se requieren habilidades bajas en DevOps. * Configuración fácil. * Complementos para casi todo. * Apto para principiantes. * Mejor para startups. * Curva de aprendizaje baja. * Alojamiento Postgres. * Fácil de agregar colaboradores. * Desarrollo más rápido. * Impresionante documentación. * Céntrese en el producto, no en la implementación. * Retroceso simple. * Fácil integración. | * Aplicaciones web. * Configuración fácil. * Superfácil de usar. * Acceso a Shell. * Gran apoyo. * Plan gratuito. * Muchas cosas como Python están preinstaladas. |

| **Desventajas Heroku** | **Desventajas Pythonanywhere** |
| --- | --- |
| * No mucha flexibilidad. * No hay opción de MySQL utilizable. * Almacenamiento. * Bajo rendimiento en el nivel gratuito. * El soporte 24/7 cuesta $ 1,000 por mes | * Sin acceso de root. * Comunidad realmente pequeña. |

**El secreto del programador**

**Otras alternativas**

Hay muchos otros servicios web semejantes 😃

Aquí los diez más populares: [Deploy Web Django](https://www.netguru.com/blog/django-hosting)

De esa lista la alternativa más completa es la [nube de Amazon](https://aws.amazon.com/es/getting-started/hands-on/deploy-python-application/)

**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.

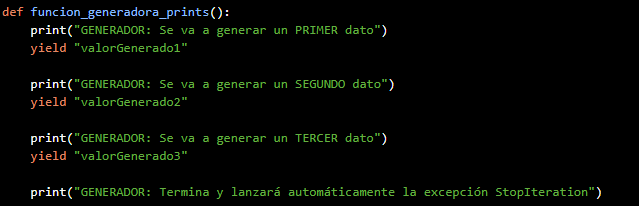
Una característica importante es que tanto las variables locales como el punto de inicio de la ejecución se guardan automáticamente entre las llamadas sucesivas que se hagan al generador.

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).

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

**Generadores**

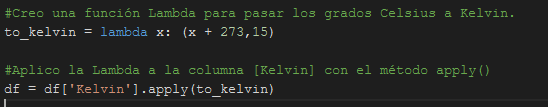
Varios ejemplos para empezar: [Generadores](https://www.codeboxsystems.com/tutorials/sp/python-yield-generators-and-generator-expressions/)



**Lambda**

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

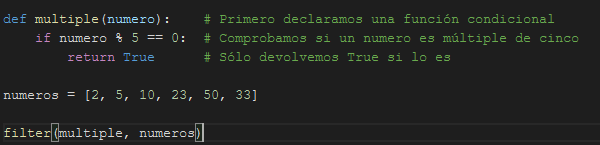
Al contrario que una función normal, no la definimos con la palabra clave estándar def que utilizamos en Python. En su lugar, las funciones Lambda se definen como una línea que ejecuta una sola expresión.



**Filter ()**

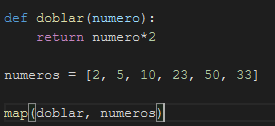
Como su nombre indica filter significa filtrar, ya que a partir de una lista o iterador y una función condicional, es capaz de devolver una nueva colección con los elementos filtrados que cumplan la condición.

Supongamos que tenemos una lista de 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:



**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.

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**

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

**Data Literacy**

Hace referencia a la “Alfabetización en datos” es decir, adquirir las capacidades de leer, comprender, crear y saber comunicar nuestros datos de manera efectiva.

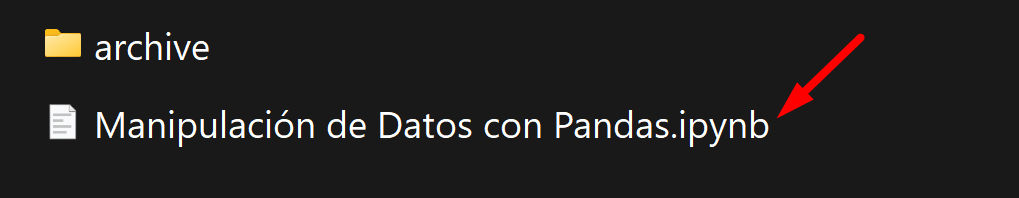
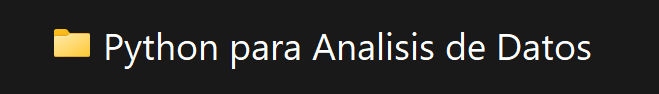
**Una nueva disciplina en auge: La “Ciencia de Datos”**

“Con el fenómeno creciente del Big Data, surge una nueva disciplina conocida como Data Science o Ciencias de Datos. De esta intersección, se desprende una figura totalmente disruptiva que mezcla conocimientos de estadística, matemáticas, management empresarial e ingeniería informática: el Científico de Datos.”

**Empresas Data Driven**

“Las empresas Data - Driven son aquellas que permiten que sean exclusivamente los “datos” quienes las gobiernen. La filosofía de trabajo y cultura Data Driven, es verdaderamente un cambio de mentalidad, que se apoya en los datos para tomar mejores decisiones y escoger qué alternativa es la más óptima a elegir. Dicho con otras palabras, el “dato” en este tipo de empresas es el REY.”

**Veamos cómo podemos utilizar Python para analizar datos**

****

**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](https://www.coderhouse.com/online/desarrollo-web-online)

**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.

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.

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

Por ejemplo en este curso de CoderHouse: [SQL](https://www.coderhouse.com/online/sql)

**Recomendaciones finales**

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