

Archivos de Configuración



Introducción

Bienvenidos al módulo de Entornos virtuales en Python. Los objetivos de aprendizaje estarán orientados a incorporar conocimientos que nos permitirán trabajar con las variables de entorno en proyectos Python y utilizar correctamente la librería “decouple”.

Una vez finalizado este módulo serás capaz de:

- Crear archivos de configuración .env y .ini.
- Leer archivos .env con librería decouple.
- Utilizar placeholders en los archivos de configuración.
- Leer archivos .env con librería dotenv.



Tema 1. Variables de Entorno

Objetivo

El objetivo de este tema es brindar una descripción conceptual de las variables de entorno en proyectos Python y las ventajas de su utilización a través del archivo “.env”

Una vez finalizado este tema serás capaz de:

- Conocer el concepto de variable de entorno y su funcionalidad.
- Entender la funcionalidad del archivo “.env”

Conceptos iniciales

Una de las etapas de nuestras aplicaciones es la de Deployment. En algún momento durante el desarrollo, debemos pensar en el entorno en el cual se ejecutará la aplicación y la información potencialmente confidencial o específica del entorno que se necesitará para ejecutar.

Las variables de entorno son una de las formas clave en que los desarrolladores proporcionan una aplicación con este tipo de información, sin embargo, ejecutar y probar una aplicación localmente que depende de las variables de entorno puede ser una molestia si se están configurando esas variables en el entorno local.

El proceso de establecer o cambiar una variable de entorno requiere mucho tiempo y, con el tiempo, la cantidad de variables de entorno que se tienen que administrar crece rápidamente. Eventualmente, los conflictos de nombres se convierten en un problema y cada variable nueva requiere un prefijo largo para distinguirse de otras variables similares.

El uso de un archivo “.env” permite usar variables de entorno para el desarrollo local sin contaminar el espacio de nombres del entorno global. También permitirá mantener los nombres y valores de las variables de entorno aislados en el mismo proyecto que los utiliza.

Archivo .env

Como mencionamos anteriormente, un archivo “.env” es un archivo de texto en el cual se definen un conjunto de variables de entorno a las cuales les asignamos un valor. El mismo debe agregarse en el directorio raíz de un proyecto.

Este archivo es utilizado cuando queremos aplicar buenas prácticas a nuestro código, ya que no es una buena idea guardar este tipo de datos en scripts. El archivo “.env” contiene pares de valores clave de todas las variables de entorno requeridas por nuestra aplicación. Este archivo se incluye localmente con el proyecto, pero no debe guardarse en el control de código fuente para que no ponga en riesgo información potencialmente confidencial. En este sentido, por ejemplo, podríamos estar trabajando con los nombres y URL de bases de datos SQL, contraseñas, otras claves secretas asociadas con grupos de usuarios de AWS o Google Cloud Platform, APIS y otros. Es por esto que no quisiéramos exponer esa información en un repositorio.

```
DEBUG=True
TEMPLATE_DEBUG=True
SECRET_KEY=ARANDOMSECRETKEY
DATABASE_URL=mysql://myuser:mypassword@myhost/mydatabase
PERCENTILE=90%
#COMMENTED=42
```

Ejemplo de archivo “.env”

Archivo .INI

Los archivos .ini son quizás los archivos de configuración más directos disponibles. Los archivos .ini son muy adecuados para proyectos pequeños, principalmente porque estos archivos solo admiten jerarquías de 1 nivel de profundidad. Los archivos .ini son esencialmente archivos de texto plano, con la excepción de que las variables pueden pertenecer a grupos.

Esta estructura seguramente hace que las cosas sean más fáciles de entender, pero la practicidad de esta estructura va más allá de la estética.

```
[settings]
DEBUG=<s>True</s>
TEMPLATE_DEBUG=<s>%(DEBUG)s</s>
SECRET_KEY=<s>ARANDOMSECRETKEY</s>
DATABASE_URL=<s>mysql://myuser:mypassword@myhost/mydatabase</s>
PERCENTILE=<s>90%%</s>
#COMMENTED=42
```

Ejemplo de archivo “.ini”



Tema 2. Visual Studio Code

Objetivo

El objetivo de este tema es brindar una breve descripción de la herramienta Visual Studio Code y como instalar la misma para poder utilizarla a lo largo del proceso de aprendizaje.

Una vez finalizado este tema serás capaz de:

- Entender para que usar Visual Studio Code
- Instalar la misma

Descripción de la Herramienta

Según Wikipedia, “Visual Studio Code es un editor de código desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto, aunque la descarga oficial está bajo software privativo e incluye características personalizadas por Microsoft.

Visual Studio Code se basa en Electron, un framework que se utiliza para implementar Chromium y Node.js como aplicaciones para escritorio, que se ejecuta en el motor de diseño Blink. Aunque utiliza el framework Electron, el software no usa Atom y en su lugar emplea el mismo componente editor (Monaco) utilizado en Visual Studio Team Services (anteriormente llamado Visual Studio Online).”

De acuerdo a Microsoft: “VSC es un Editor de código fuente independiente que se ejecuta en Windows, macOS y Linux. La elección principal para desarrolladores web y JavaScript, con extensiones para admitir casi cualquier lenguaje de programación.”

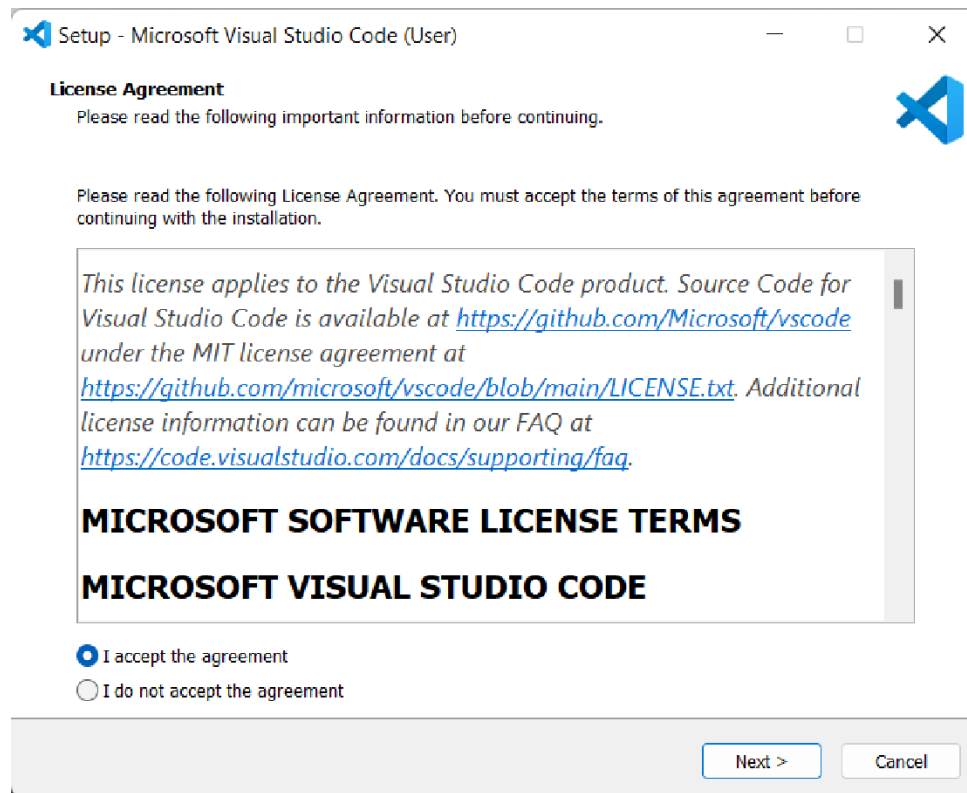
VSC es una de las herramientas mas utilizadas para el desarrollo de Software, es por eso que utilizaremos la misma para trabajar en el resto del documento.

Instalación

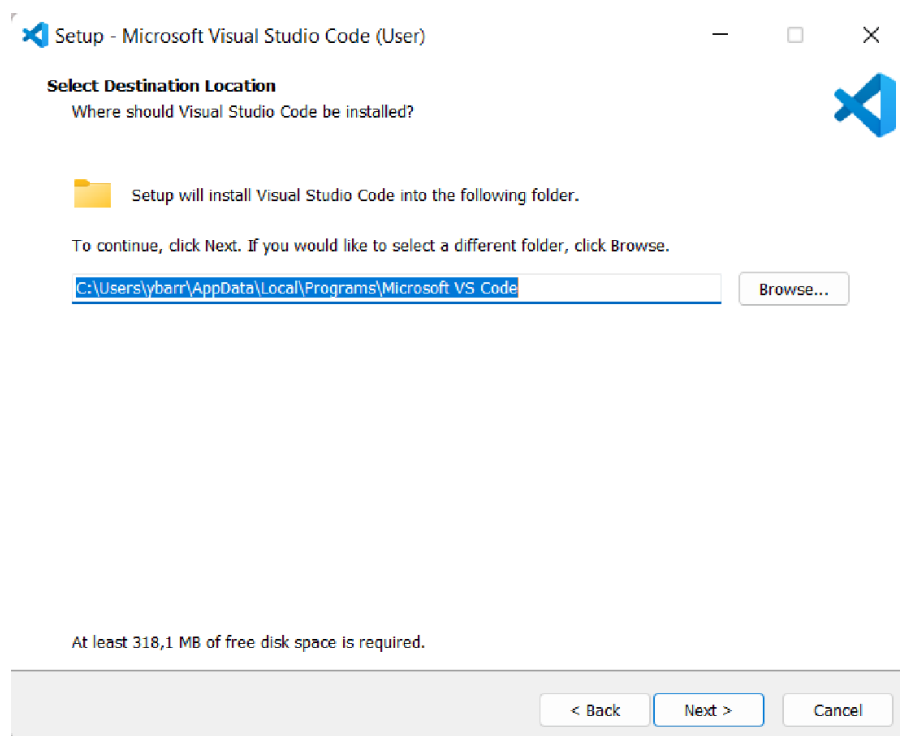
Descargamos el instalador para nuestro SO desde el siguiente link:

<https://code.visualstudio.com/>

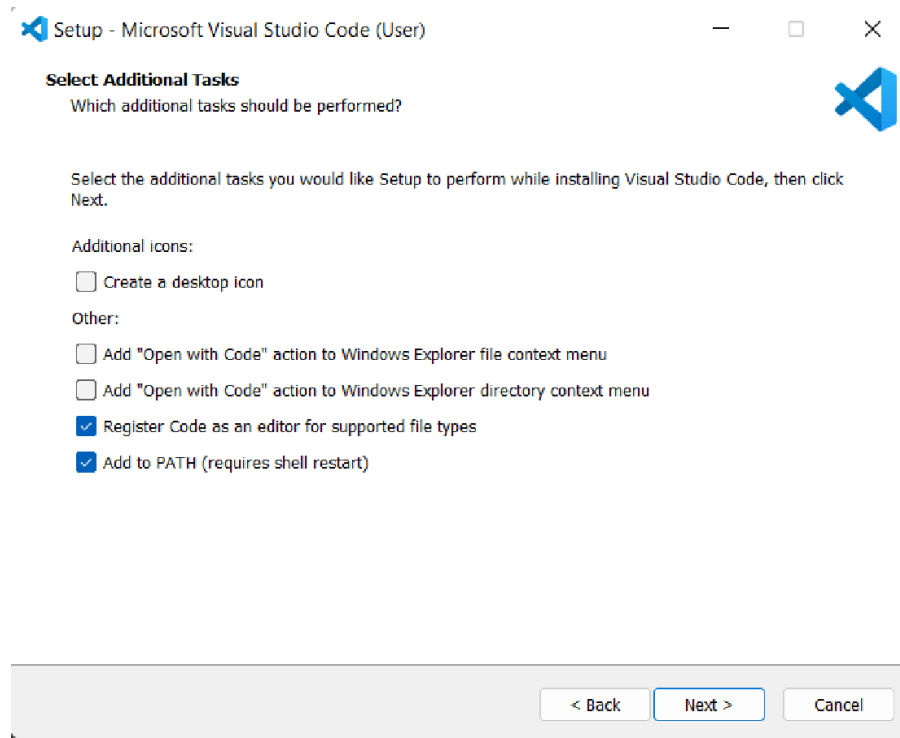
Ejecutamos el mismo y nos aparece la siguiente pantalla:



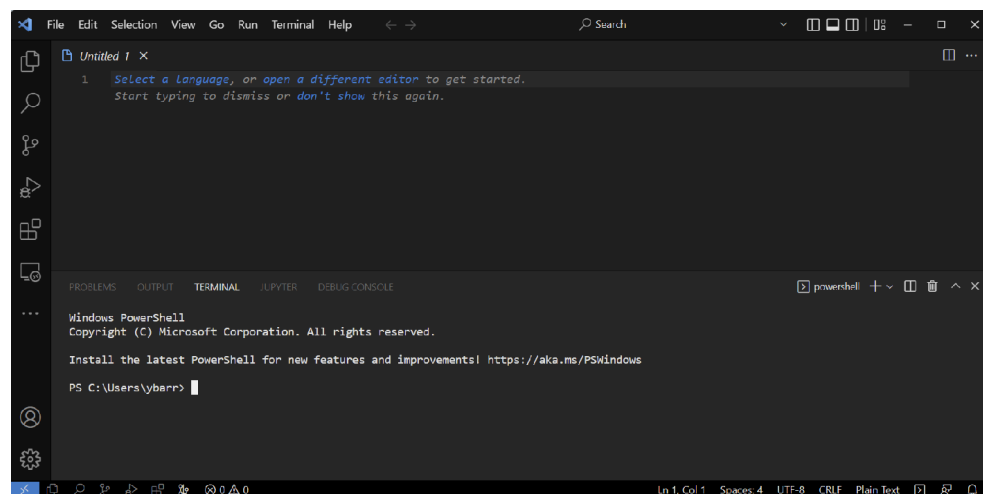
Aceptamos los términos del licenciamiento y presionamos “next”. El sistema nos mostrará la siguiente pantalla:



Aceptamos (O modificamos según necesidad) la ruta de instalación y presionamos “Next”. El sistema nos mostrará la siguiente pantalla:



Dejamos los valores por defecto y presionamos “Next”. El sistema comenzará con la instalación. Al finalizar presionamos “Launch”. Nos aparecerá la pantalla de Visual Studio Code, donde podremos comenzar a trabajar.



Una vez finalizada la instalación, podemos avanzar para poder trabajar con las librerías que nos permiten acceder a las variables de entorno en nuestros proyectos Python.



Tema 3. Decouple

Objetivo

El objetivo de este tema es brindar una descripción básica del funcionamiento de la librería Decouple.

Una vez finalizado este tema serás capaz de:

- Instalar Decouple en un ambiente virtual
- Leer datos variables de entorno de un archivo .env

Definición

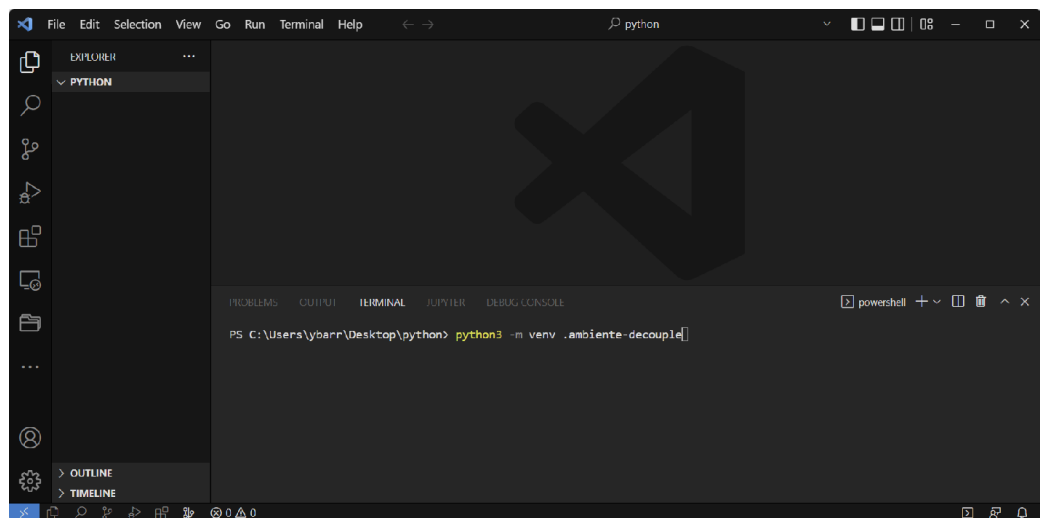
De acuerdo a la descripción del sitio oficial, “Decouple” ayuda a organizar la configuración de una aplicación para que se puedan cambiar los parámetros sin tener que volver a implementar la misma.

También facilita:

- Almacenar parámetros en archivos .ini o .env
- Definir valores predeterminados completos
- Convertir correctamente los valores al tipo de datos correcto;
- Tener un solo módulo de configuración para administrar todas las instancias.

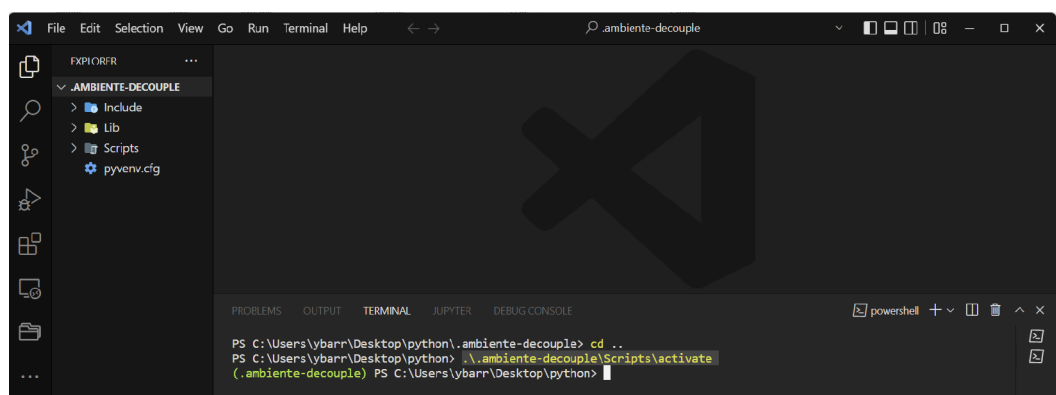
Instalación

Primero ejecutaremos visual studio code y abriremos la carpeta de nuestro proyecto. En este caso esta vacía y se llama Python. Abrimos una consola desde el menú de VSC y ejecutamos el comando “python3 -m venv .ambiente-decouple” para crear un ambiente llamado “Ambiente-decouple” en nuestro proyecto.



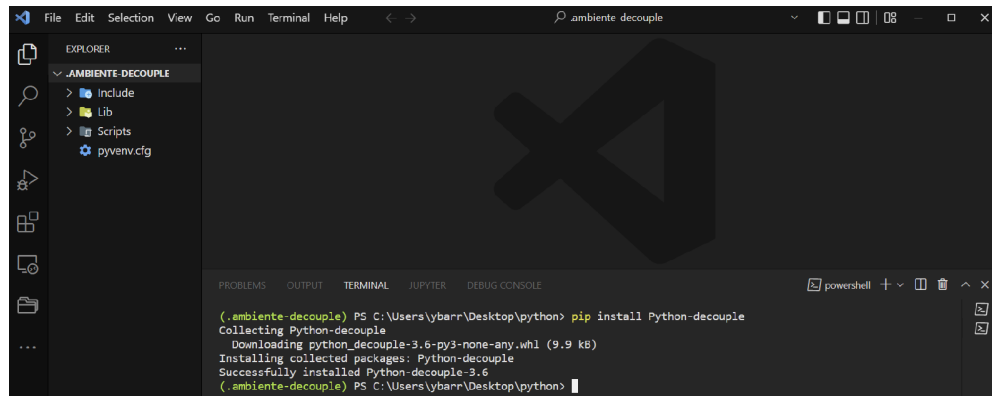
Una vez creado el ambiente, debemos activarlo. Posicionados en la carpeta de nuestro proyecto, ejecutamos el siguiente comando:
`.\ambiente-decouple\Scripts\activate`

Ahora VSC nos mostrara el prompt de la siguiente forma:



Una vez activo el ambiente virtual comenzaremos con la instalación de decouple en el mismo.

Desde la consola de VSC y mediante el siguiente comando instalaremos Decouple: `pip install Python-decouple`

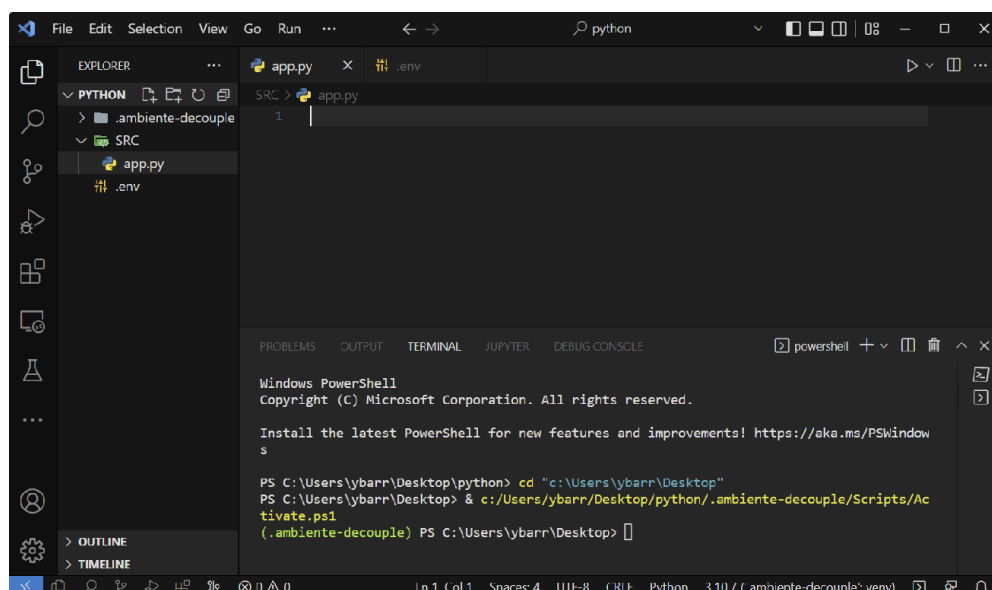


```

(.ambiente-decouple) PS C:\Users\ybarr\Desktop\python> pip install Python-decouple
Collecting Python-decouple
  Downloading python_decouple-3.6-py3-none-any.whl (9.9 kB)
Installing collected packages: Python-decouple
Successfully installed Python-decouple-3.6
(.ambiente-decouple) PS C:\Users\ybarr\Desktop\python>
  
```

VSC nos informa que fue instalado correctamente. Ahora procedemos a trabajar con la misma.

Primero crearemos una carpeta SRC fuera del ambiente virtual, y dentro de la misma crearemos un archivo `app.py`. Además, creamos un archivo `.env` al mismo nivel que SRC y el ambiente virtual. La estructura debe quedarnos según la siguiente imagen:



```

EXPLORER
├── PYTHON
│   ├── .ambiente-decouple
│   │   ├── SRC
│   │   │   ├── app.py
│   │   │   └── .env
  
```

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\ybarr\Desktop\python> cd "c:\Users\ybarr\Desktop"
PS C:\Users\ybarr\Desktop> & c:\Users\ybarr\Desktop\python\.ambiente-decouple\Scripts\Activate.ps1
(.ambiente-decouple) PS C:\Users\ybarr\Desktop>
  
```

Dentro del archivo `“.env”` cargamos los siguiente datos, los cuales necesitamos acceder desde nuestra aplicación:

```
POSTGRESQL_HOST=localhost  
POSTGRESQL_PORT=5432  
POSTGRESQL_USER=MyUser  
POSTGRESQL_PWD=MyPasswordSecreta
```

Guardamos los cambios y procedemos a trabajar con nuestra app para poder acceder a esas variables. Para eso escribimos el siguiente código en el archivo app.py:

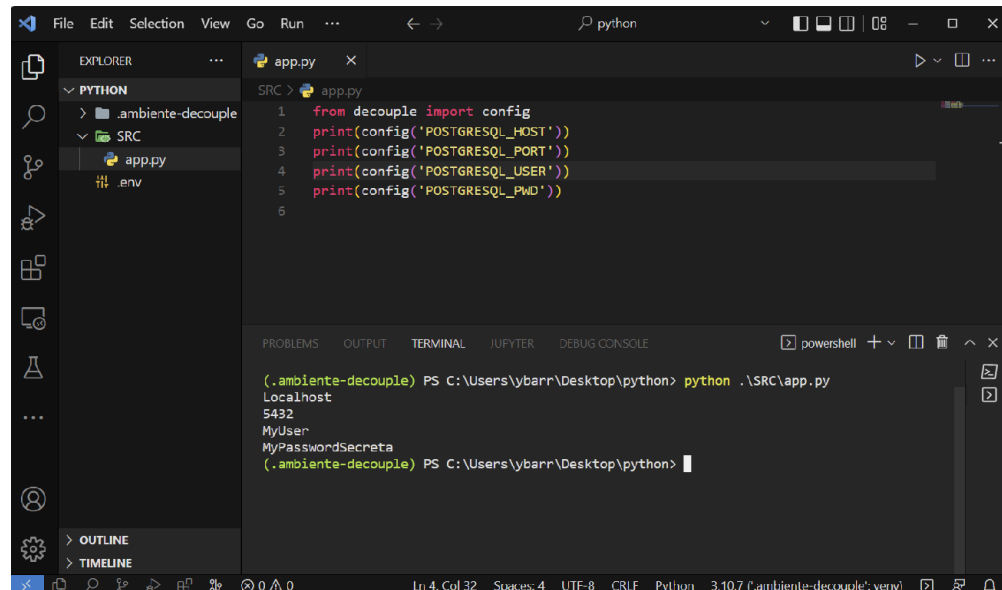
```
from decouple import config  
print(config('POSTGRESQL_HOST'))  
print(config('POSTGRESQL_PORT'))  
print(config('POSTGRESQL_USER'))  
print(config('POSTGRESQL_PWD'))
```

Guardamos los cambios.

Ahora bien, analicemos un poco nuestro código. En la primera línea importamos desde decouple, un objeto llamado “config”. Luego, en las siguientes líneas imprimimos cada uno de los valores de las 4 propiedades indicadas para el objeto “config”.

Entonces, ejecutamos nuestra aplicación con el siguiente código:
python .\SRC\app.py

VSC nos mostrara la siguiente pantalla donde se imprimen por pantalla los valores de las variables de entorno definidas en el archivo “.env”



The screenshot shows a Visual Studio Code editor with a file explorer on the left showing a project structure with a folder named 'PYTHON' containing a file 'app.py'. The main editor window displays the content of 'app.py', which is a Python script using the 'decouple' library to read environment variables. The script prints the values of 'POSTGRES_HOST', 'POSTGRES_PORT', 'POSTGRES_USER', and 'POSTGRES_PWD'. Below the editor, the 'TERMINAL' panel shows the command 'python .\SRC\app.py' being executed, resulting in the following output:

```
(.ambiente-decouple) PS C:\Users\ybarr\Desktop\python> python .\SRC\app.py
localhost
5432
MyUser
MyPasswordSecreta
(.ambiente-decouple) PS C:\Users\ybarr\Desktop\python>
```

Cabe recalcar que no debemos hacer un seguimiento a este archivo cuando trabajemos con Git, de forma tal que no se suba al repositorio debido a que el mismo posee datos sensibles.

Utilizando Placeholders

¿Qué es un placeholder? En Python, es una palabra, caracteres o una cadena de caracteres para mantener un lugar temporal.

El Placeholder se comporta como un marcador de posición dinámico de tal manera que puede pasar un valor particular para ese marcador de posición. En Python los placeholders se definen con los caracteres "{}".

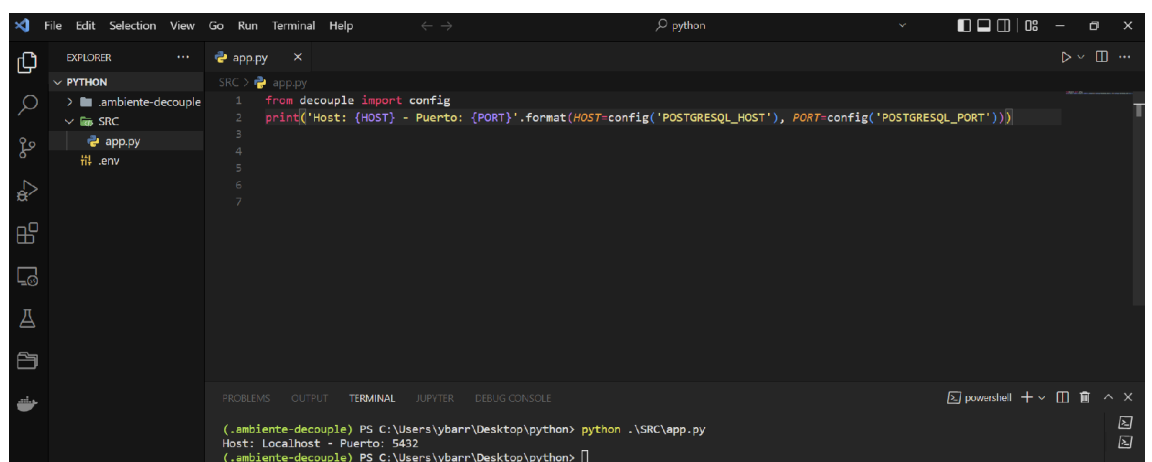
Veamos un ejemplo sencillo con nuestras variables de entorno que aprendimos a leer. Modifiquemos nuestro código por el siguiente:

```
from decouple import config
print('Host: {HOST} - Puerto:
{PORT}'.format(HOST=config('POSTGRES_HOST'),
PORT=config('POSTGRES_PORT')))
```

Ahora bien, analicemos el mismo.

La primera línea fue explicada anteriormente.

En la segunda línea estamos utilizando placeholders (En este caso son {HOST} y {PORT}) para poder completar los mismos con los valores que tienen las propiedades del objeto config, obtenido del archivo.env, con las variables de entorno.



The screenshot shows a VS Code editor with a Python file named `app.py` open. The code in the file is:

```
1 from decouple import config
2 print('Host: {HOST} - Puerto: {PORT}'.format(HOST=config('POSTGRES_HOST'), PORT=config('POSTGRES_PORT')))
3
4
5
6
7
```

The Explorer sidebar on the left shows the project structure with a folder named `.ambiente-decouple` containing `app.py` and `.env`. The bottom panel shows the terminal output:

```
(.ambiente-decouple) PS C:\Users\ybarr\Desktop\python> python .\SRC\app.py
Host: localhost - Puerto: 5432
(.ambiente-decouple) PS C:\Users\ybarr\Desktop\python>
```

Ejecutamos el código. Una vez ejecutado, el sistema nos muestra las variables de entorno utilizando los placeholders definidos.



Tema 3. Librería DotEnv

Objetivo

El objetivo de este tema es brindar una descripción de la librería `dotenv`, la cual se utiliza para manejar las variables de entorno en un proyecto Python.

Una vez finalizado este tema serás capaz de:

- Entender el funcionamiento de la librería `dotenv`

- Instalar la librería para un proyecto python
- Acceder a variables de entorno con DotEnv
- Entender los conceptos básicos del manifiesto de los 12 factores.

Definición

La librería DotEnv es una dependencia de Flask, la cual se encarga de controlar los pares de clave-valor de un archivo que sea etiquetado como archivo de entorno (“.env”).

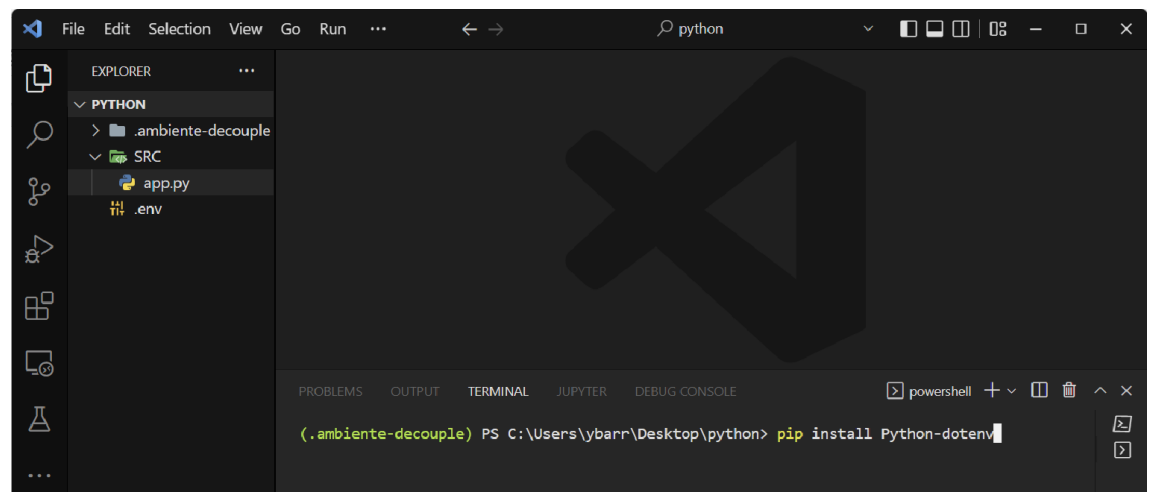
Entonces, ¿qué es Flask? Flask es un microframework escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código.

Volviendo a enfocarnos en la librería “dotenv” y de acuerdo a la documentación oficial de la misma, se indica que “dotenv”: “Ayuda en el desarrollo de aplicaciones siguiendo los principios de los 12 factores.”

Pero... ¿Cuáles son los 12 factores? Lo veremos al final de este tema

Instalación de DotEnv

Para poder instalar dotenv ejecutaremos el siguiente comando: `pip install python-dotenv`



Una vez instalado, utilizaremos los archivos creados en el tema anterior. Entonces, para acceder a las variables de entorno con dotenv, modificamos nuestro archivo app.py y agregamos el siguiente código:

```
import os
from dotenv import load_dotenv, find_dotenv
load_dotenv (find_dotenv())
print(os.getenv('POSTGRES_HOST'))
print(os.getenv('POSTGRES_PORT'))
print(os.getenv('POSTGRES_USER'))
print(os.getenv('POSTGRES_PASSWORD'))
```

Guardamos los cambios.

Ahora bien, analicemos el código escrito:

En la primera línea importamos el módulo OS. El módulo OS nos permite acceder a funcionalidades dependientes del Sistema Operativo. Sobre todo, aquellas que nos refieren información sobre el entorno del mismo y nos permiten manipular la estructura de directorios.

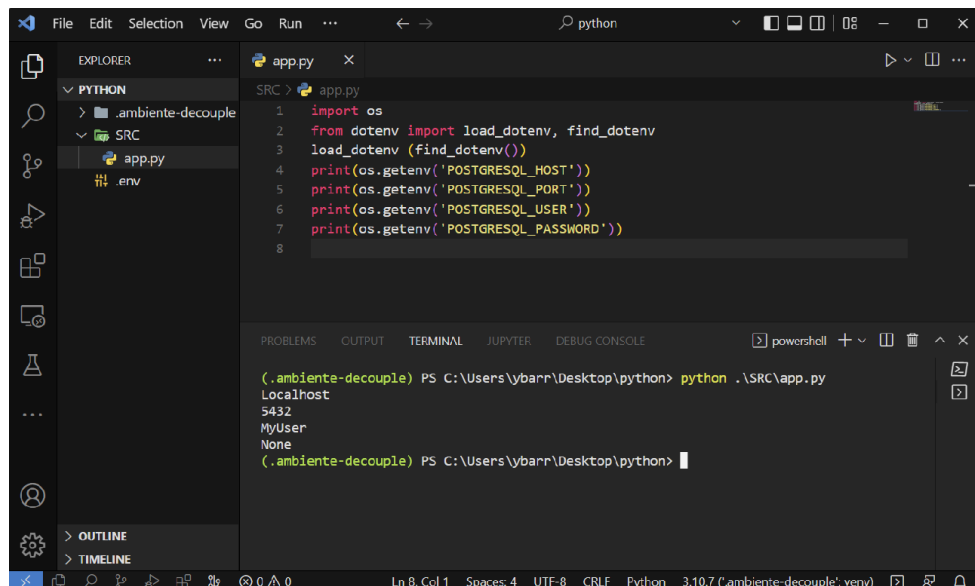
En la segunda línea importamos load_dotenv y find_dotenv. Load_dotenv se utilizara para tomar las variables de entorno del

archivo de configuración. Luego, find_dotenv se utilizara para ubicar el archivo .env en nuestro proyecto.

En la tercera línea, utilizamos los find_dotenv y load_dotenv.

Finalmente, en las siguientes líneas, solicitamos imprimir en pantalla las variables de entorno almacenadas en nuestro archivo .env.

Luego, ejecutamos nuestra aplicación.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane shows a project structure with a folder named 'PYTHON' containing a subfolder 'ambiente-decouple' and a file 'app.py'. The main editor displays the code in 'app.py':

```
1 import os
2 from dotenv import load_dotenv, find_dotenv
3 load_dotenv(find_dotenv())
4 print(os.getenv('POSTGRES_HOST'))
5 print(os.getenv('POSTGRES_PORT'))
6 print(os.getenv('POSTGRES_USER'))
7 print(os.getenv('POSTGRES_PASSWORD'))
8
```

Below the editor, the TERMINAL pane shows the command prompt output after running the script:

```
(.ambiente-decouple) PS C:\Users\ybarr\Desktop\python> python .\SRC\app.py
Localhost
5432
MyUser
None
(.ambiente-decouple) PS C:\Users\ybarr\Desktop\python>
```

VSC nos mostrara la siguiente pantalla donde se imprimen por pantalla los valores de las variables de entorno definidas en el archivo “.env” pero en este caso hemos utilizado la librería dotenv.

Los 12 Factores

Anteriormente en este documento hemos mencionado el concepto de los 12 factores. Pero... que son los 12 factores? El manifiesto de los 12 factores contiene los puntos fundamentales que debería cumplir el

desarrollo de una aplicación preparada para ser distribuida como un servicio y se desarrolla en los siguientes puntos:

- 1- **Código Base:** el código debe estar en repositorios y es desde ahí donde se harán los despliegues a los entornos. Puede haber varios repositorios si la aplicación es compleja, pero no se puede repetir código entre repositorios.
- 2- **Dependencias:** todas las dependencias de la aplicación deben ser explícitas y existir un sistema automático que las instale si no existen.
- 3- **Configuraciones:** el código no puede tener configuraciones, sino que deben ser separadas en otros recursos dependientes del entorno. Así, el código debe hacer referencia a las variables de entorno pero no a sus valores. Esto permite instalar el mismo código en diferentes entornos.
- 4- **Backing services:** son recursos conectables a la aplicación, independientemente de si son locales o de terceros. Si la aplicación necesita una base de datos, será tratada como un recurso independiente, con su URL y su instalación, que deben ser proporcionados a la aplicación. Si un servidor de bases de datos proporciona 2 esquemas diferentes, serán tratados como si estuvieran en servidores diferentes.
- 5- **Construir, distribuir y ejecutar:** son las 3 etapas que se deben pasar para transformar el código fuente en un producto utilizable. Deben ser tratadas de forma independiente y con un control de las versiones que se van distribuyendo acorde a un sistema preestablecido.
- 6- **Procesos sin estado (stateless):** la aplicación se ejecuta en el destino como un proceso (o varios) en la máquina anfitriona pero la información que existe en su espacio de memoria sólo tiene lo

necesario para hacer las transacciones que lleva a cabo en ese momento, no para atender a transacciones futuras.

- 7- **Asignación de puertos:** el código no puede considerar a priori la asignación de los puertos. Simplemente sabe que es un proceso que escucha en un puerto arbitrario. Será la configuración la que establezca los puertos de cada componente.
- 8- **Concurrencia:** las aplicaciones deben estar desarrolladas teniendo en mente la concurrencia, bien de hilos o bien de procesos. La razón principal es la naturaleza distribuida y la capacidad para escalar levantando nuevas máquinas.
- 9- **Disponibilidad:** las aplicaciones deben ser capaces de iniciar sus procesos rápidamente y de responder de forma controlada a la finalización, aunque esta sea inesperada.
- 10- **Paridad en el desarrollo y producción:** se refiere a tener los diferentes entornos lo más similares posibles, tanto de versiones de código desplegado, como de entornos y servicios.
- 11- **Historiales:** o más conocidos como «Logs». Las aplicaciones no se preocupan por el direccionamiento y almacenamiento de sus datos de logs. Para eso existen sistemas específicos en cada entorno.
- 12- **Administrador de procesos:** además de los procesos propios de la ejecución de la aplicación, existirán procesos de tareas de administración y mantenimiento que suelen ejecutarse una única vez. Estos procesos, aunque puedan parecer secundarios, se ejecutarán con igual importancia que el resto de los procesos de la aplicación



Referencias

<https://dev.to/jakewitcher/using-env-files-for-environment-variables-in-python-applications-55a1>

<https://code.visualstudio.com/>

<https://www.adictosaltrabajo.com/2016/08/30/el-manifiesto-the-twelve-factor-app/>

<https://12factor.net/es/>

<https://pypi.org/project/python-dotenv/>

tiiiiiit by 