1000 PROGRAMADORES

Introducción a la
Programación en Python

MÓDULO 8

Bases de Datos.
Git.
Metodologías ágiles de desarrollo.

Ministerio de Educación
Cultura, Ciencia y Tecnología
Gobierno de Salta

Ministerio de Economía
y Servicios Públicos
Gobierno de Salta

Universidad
Nacional de Salta

INTRODUCCIÓN

También conocidas como bancos de datos, son simplemente conjuntos de datos que serán utilizados a través de una o más tareas que producen la información necesaria por un individuo u organización.

Este conjunto de datos puede ser de cualquier tipo, como números, cadenas de caracteres, las fechas, etc. que hacen referencia a información perteneciente a un mismo contexto. Es decir, si hablamos de los clientes de una empresa, éstos datos comunes de todos los clientes podrían ser sus nombres, apellidos, direcciones, teléfonos. Si fuesen los productos de una tienda, cada uno con su nombre, precio, descripción, etc.

Por tanto, al ser datos del mismo contexto, las bases de datos permiten almacenarlos sistemáticamente, de forma automatizada, con la finalidad de un posterior uso. Ya sea para realizar consultas, comparativas, análisis, modificaciones o simplemente borrarlos.

Podríamos verlo así también, las palabras son datos, un libro es información porque tiene datos (estas palabras) ordenadas con un propósito (contar una historia) y una biblioteca es una base de datos porque almacena esa información de forma ordenada, por ejemplo por autor.

Quienes se encargan del orden de una base de datos son programas complejos centrados en la gestión de información y reciben el nombre de SGBD: Sistemas Gestores de Bases de Datos

La peculiaridad de los SGBD es que implementan sus propios lenguajes internos de programación par realizar las consultas, que pueden ser tan simples como consultar el nombre de todos los clientes de la empresa, o tan complejas como conseguir información de los pedidos de una tienda, a la vez que se consultan los clientes y en un conjunto de fechas determinado.

MODELOS DE BASE DE DATOS

Los tipos de datos y la forma de almacenarlos pueden diferir mucho dependiendo del contexto, con el tiempo se han ido desarrollando una serie de modelos distintos para gestionar las bases de datos.

- ➔ Jerárquicas
- ➔ De red
- ➔ Transaccionales
- ➔ Documentales
- ➔ Orientadas a objetos
- ➔ Deductivas
- ➔ Relacionales

Modelo Relacional

Es utilizado en la actualidad para representar problemas reales y administrar datos dinámicamente gracias a que es fácil representar y gestionar problemas del mundo real. Es en el que nos vamos a centrar.

Las bases de datos Relacionales son muy utilizadas actualmente. Se basan en la idea de crear relaciones entre conjuntos de datos, en los que cada relación es también una tabla. Cada tabla consta de registros, formados por filas y columnas, también conocidos como tuplas y campos.

Dentro de las bases de datos relacionales, existen muchos SGBD. La mayoría son compatibles con Python. Algunos son de pago, otros gratuitos, los hay sencillos y otros muy avanzados. Hagamos un repaso:

- ➢ SQL Server
- ➢ Oracle
- ➢ MySQL:
- ➢ PostgreSQL
- ➢ SQLite

En Python, cada uno de ellos cuenta con módulos libres y programas conectores para comunicar las bases de datos y el lenguaje de programación. Sin embargo, pese a que son sistemas distintos, el lenguaje de las consultas no varía mucho, sino sería muy difícil pasar de un sistema a otro y los SGBD no podrían competir entre ellos.

Lo que nos lleva a nuestra última cuestión, ¿qué tienen en común? ¿Qué son esas siglas SQL en sus nombres?

El lenguaje SQL

Aparte de los lenguajes de programación como Python, centrados en la creación de los programas, los SGBD implementan su propia sintaxis o lenguaje propio para realizar consultas y modificaciones en sus registros.

El lenguaje más utilizado en las bases de datos relacionales es el lenguaje SQL (Lenguaje de Consulta Estructurada), y es necesario aprenderlo si queremos utilizar este tipo de bases de datos en nuestros programas.

Este lenguaje abarca muchísimo contenido, por lo que en este módulo sólo veremos algunas consultas básicas para utilizar el SQLite en nuestros scripts de Python.

BASES DE DATOS EN PYTHON

Los pasos a seguir para conectar y usar una Base de Datos son:

1. Abrir/Crear la conexión
2. Crear puntero
3. Ejecutar una consulta SQL (query)
4. Administrar los resultados de la consulta
    - Insertar (Create)
    - Leer (Read)
    - Actualizar (Update)
    - Borrar (Delete)
5. Cerrar puntero
6. Cerrar conexión

Conexión a la base de datos, creación y desconexión

Al realizar la conexión, si la base de datos no existe, entonces la crea. Siempre debe cerrarse la conexión al finalizar el uso, sino luego no se podrá seguir manipulandola.

```
import sqlite3                                  # Importamos el módulo

conexion = sqlite3.connect('ejemplo.db')  # Conexión a la BD

conexion.close()                                # Cerramos la conexión
```

Crear una tabla

Antes de ejecutar una consulta (query) en código SQL, tenemos que crear un cursor:

```
import sqlite3

conexion = sqlite3.connect('ejemplo.db')

# Creamos el cursor
cursor = conexion.cursor()

# Crearemos una tabla de usuarios con nombres, edades y emails
cursor.execute("CREATE TABLE IF NOT EXISTS usuarios (nombre
VARCHAR(100), edad INTEGER, email VARCHAR(100))")

conexion.close()
```

Inserción o carga de datos en una tabla

Cargar una tupla

```
import sqlite3

conexion = sqlite3.connect('ejemplo.db')

cursor = conexion.cursor()

# Insertamos un registro en la tabla de usuarios
cursor.execute("INSERT INTO usuarios VALUES ('Hector', 27, 'hector@ejemplo.com')")

# Guardamos los cambios haciendo un commit
conexion.commit()

conexion.close()
```

Cargar varias tuplas

Insertando varios registros con *.executemany()* :

```
import sqlite3

conexion = sqlite3.connect('ejemplo.db')
cursor = conexion.cursor()

# Creamos una lista con varios usuarios
usuarios = [('Mario', 51, 'mario@ejemplo.com'),
            ('Mercedes', 38, 'mercedes@ejemplo.com'),
            ('Juan', 19, 'juan@ejemplo.com')]

# Ahora utilizamos el método executemany() para insertar varios
cursor.executemany("INSERT INTO usuarios VALUES (?,?,?)", usuarios)

# Guardamos los cambios haciendo un commit
conexion.commit()
conexion.close()
```

Lectura de datos en una tabla

Lectura de una tupla

Recuperando el primer registro con *.fetchone():*

```
import sqlite3

conexion = sqlite3.connect('ejemplo.db')
cursor = conexion.cursor()

# Recuperamos los registros de la tabla de usuarios
cursor.execute("SELECT * FROM usuarios")

# Recorremos el 1er registro con fetchone, devuelve una tupla
usuario = cursor.fetchone()
print(usuario)

conexion.close()
```

Lectura múltiple

Recuperando varios registros con .fetchall():

```
import sqlite3

conexion = sqlite3.connect('ejemplo.db')
cursor = conexion.cursor()

# Recuperamos los registros de la tabla de usuarios
cursor.execute("SELECT * FROM usuarios")

# Se recorren los registros con fetchall y se vuelcan a una lista
usuarios = cursor.fetchall()

# Ahora podemos recorrer todos los usuarios
for usuario in usuarios:
   print(usuario)

conexion.close()
```

GIT

Es un software de control de versiones distribuido para desarrolladores, de código abierto y gratuito diseñado para manejar todo, desde proyectos pequeños a muy grandes, con velocidad y eficiencia.

El control de versiones se refiere al proceso de guardar diferentes archivos o «versiones» a lo largo de las diferentes etapas de un proyecto. Esto permite a los desarrolladores hacer un seguimiento de lo que se ha hecho y volver a una fase anterior si deciden que quieren revertir algunos de los cambios que han hecho.

Esto es útil por varias razones. Por ejemplo, facilita la resolución de errores y la corrección de otros errores que puedan ocurrir durante el desarrollo. También puede anotar los cambios en cada versión, para ayudar a cualquier miembro del equipo a mantenerse al día sobre lo que se ha completado y lo que aún queda por hacer.

En el mundo del pasado la gente manejaba las versiones de archivos con nombres como *“Versión Final”*, *“Versión Final 2.0”*, *“Versión final final final este sí”*.

Uno de los software más famosos del pasado es ***Concurrent Version System*** o ***CVS***. CVS era muy bueno y popular, abrió al mundo algo extraordinario como lo es el sistema de control de versiones de una manera distribuida y funcionaba en múltiples sistemas. Pero era difícil de entender. Para hacerlo más fácil, para tener diferentes usuarios, que fuese Open Source y demás nace ***Subversión*** o ***SVN***. Subversion no era suficiente, en especial en el mundo Open Source.

El proyecto más importante del mundo de código abierto es Linux. Creado por Linus Torvalds. Así que decidió crear desde cero su propio sistema de control de versiones con ramas y lo llamó ***Git***.

-.-

Git es un sistema donde las personas pueden usar ramas. Puedes tener tres ramas diferentes, con nombres diferentes, donde
se pueden hacer cambios diferentes a un mismo archivo y donde diferentes personas pueden estar trabajando en cosas distintas.
Luego se pueden fusionar esos cambios y mandarlos a producción en una sola rama (master).

-.-

El problema es que Git requiere que tengas conocimientos de terminal y línea de comandos. Todas las personas que programan deberían conocer sobre esto, pero puede resultar difícil si se está empezando. Hay interfaces gráficas para Windows y Mac para hacerlo un poco más amigable, pero no se había hecho algo centralizado hasta el 2012 cuando nació GitHub.

GitHub

GitHub facilita la colaboración con git. Es una plataforma que puede mantener repositorios de código en almacenamiento basado en la nube para que varios desarrolladores puedan trabajar en un solo proyecto y ver las ediciones de cada uno en tiempo real.

Incluye funciones de organización y gestión de proyectos. Puede asignar tareas a individuos o grupos, establecer permisos y roles para los colaboradores y usar la moderación de comentarios para mantener a todos en la tarea.

Los repositorios de GitHub están disponibles públicamente. Los desarrolladores de todo el mundo pueden interactuar y contribuir al código de los demás para modificarlo o mejorarlo, lo que se conoce como «codificación social». En cierto modo, esto hace que GitHub sea un sitio de redes para profesionales de la web.

Hay tres acciones principales que puede realizar cuando se trata de interactuar con el código de otros desarrolladores en GitHub:

- **Bifurcación:** El proceso de copiar el código de otra persona del repositorio para modificarlo.
- **Pull:** Cuando haya terminado de hacer cambios en el código de otra persona, puede compartirlos con el propietario original a través de una «solicitud pull».
- **Fusión:** Los propietarios pueden añadir nuevos cambios a sus proyectos a través de una fusión, y dar crédito a los contribuyentes que los han sugerido.

Especialmente para los nuevos desarrolladores que están tratando de construir sus currículums, esta puede ser una gran oportunidad para ganar algo de experiencia. GitHub le permite compartir proyectos en su perfil y mantiene una línea de tiempo de todos aquellos en los que ha contribuido.

GitLab

Años después, nace GitLab. Básicamente es GitHub, pero open source y a partir de entonces se empiezan a separar mucho más. La idea principal era tener esta herramienta pero con código abierto.

El aporte más grande de GitHub al mundo no es solo el hecho de colocarle una interfaz web al sistema de control de versiones, sino los Pull Requests. ¿Qué es esto? Es tener un código y que otra persona pueda mejorarlo agregándole tal cantidad de líneas; el dueño de ese código tiene la opción de aceptar esa propuesta o no.

Otro de los aportes de GitHub es el perfil de usuario de programador donde se refleja qué es lo que hace cada uno de los programadores que aportan a un proyecto.

METODOLOGÍAS ÁGILES DE DESARROLLO

Algunos se preguntarán qué significa eso de "ágil" y si conlleva de forma intrínseca un poco de "hacer mal las cosas" o de "dejarlo a medias". Cierto es que durante este curso casi todo lo que hemos intentado enseñar es precisamente a ser metódicos, rigurosos y estrictos científicamente hablando, aunque también hemos intentado fomentar el análisis crítico.

Ser "ágiles" no significa renunciar a formalismos ni dejar de ser estrictos y rigurosos. Muchos profesionales del tema han dicho en alguna ocasión algo como esto "eso es en teoría y queda muy bonito pero en la práctica no puedes perder tanto tiempo haciendo el análisis porque cuando acabas el sistema ya ha cambiado y el cliente se ha aburrido". Hay que confesar lo fácil que se puede caer en esta dinámica cuando la presión está encima y los plazos de entrega se acercan.

La alta competitividad actual hace que los sistemas de información se tengan que desarrollar de forma rápida para adaptarse a la organización. Las prisas hacen que lo primero que se deseche en esta carrera "loca" hacia un rápido desarrollo sea un análisis exhaustivo y se sustituye por uno superficial o simplemente se elimina. Éste es sin duda el gran error, deseamos tener un sistema desarrollado rápidamente pero con lo que realmente nos encontramos entre las manos es con un sistema lleno de errores, inmanejable y que no se puede mantener. Es difícil cambiar las reglas del mercado mundial, así que lo que se ha pensado es adaptar las metodologías de especificación y desarrollo a este entorno cambiante y lleno de presiones, en el que obtener un resultado rápido, algo que se pueda ver, mostrar y sobre todo utilizar, se ha vuelto crucial para el éxito de las organizaciones.

La metodología necesariamente debe ser ágil, debe tener un ciclo corto de desarrollo y debe incrementar las funcionalidades en cada iteración del mismo preservando las existentes, ayudando al negocio en lugar de darle la espalda. Es para este entorno que han nacido las metodologías ágiles.

Las metodologías ágiles no son la gran solución a todos los problemas del desarrollo de aplicaciones, ni tan siquiera se pueden aplicar en todos los casos, pero sí que nos aportan otro punto de vista de cómo se pueden llegar a hacer las cosas, de forma más rápida, más adaptable y sin tener que perder la rigurosidad de las metodologías clásicas.

¿Qué son las metodologías ágiles?

Son una estrategia integral que impulsa a las organizaciones a gestionar los proyectos con rapidez y flexibilidad.

La realidad es que el mercado cada día exige mayor flexibilidad ante un panorama incierto y cambiante, y las empresas deben responder con urgencia esta demanda.

"El 48% de los proyectos no se terminan dentro del tiempo planificado" – PMI

La metodología Agile ayuda en el desarrollo de proyectos que necesitan rapidez y flexibilidad para adecuarse a las necesidades del cliente. Siempre enfocada a mejorar resultados. A diferencia de la forma tradicional de gestionar los proyectos, las metodologías ágiles no necesitan definir al inicio de los proyectos la totalidad del alcance.

Es una innovadora forma de trabajar y organizarse que "fragmenta" los proyectos en partes capaces de adaptarse sobre la marcha, complementarse y resolverse en poco tiempo. Es decir, no se planifica ni se diseña el proyecto por adelantado, sino que a medida que se desarrolla se va definiendo el proyecto, gracias a un feedback constante.

Otra característica muy particular es que se trabaja por períodos de tiempo durante el cual cada miembro del equipo debe ejecutar una serie de tareas. Luego de ejecutar dichas tareas, se entregan los avances, se reciben devoluciones y comienza nuevamente el proceso, permitiendo implementar los cambios necesarios.

¿Cuándo nació el concepto "Agile"?

En el año 2001, debido a que las metodologías tradicionales no respondían a las necesidades del momento, se reunieron los principales directivos de las empresas de desarrollo de software para dialogar sobre la necesidad de crear nuevas herramientas.

Es en ese momento donde nacen oficialmente las metodologías ágiles y decimos oficialmente, porque tenemos un hecho predecesor en 1950 cuando se utilizó un método repetitivo e incremental para el desarrollo de software del cohete X-15

Además, un hito que marcaría un antes y un después, fue en 1985 con la presentación de SCRUM por Ken Schwaber.

Los 12 principios del Manifiesto Agile

Sus creadores dispusieron de 12 principios para definir la filosofía de las metodologías ágiles.

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

¿Qué ventajas tienen las metodologías ágiles?

En primer lugar es importante entender que las metodologías ágiles nacen para privilegiar la interacción en los procesos y facilitar la colaboración con el cliente ante la necesidad de cambios durante el armado del proyecto.

➔ **Entregas rápidas y continuas**

Uno de los aspectos más importantes de las metodologías ágiles es que tienen como principal característica realizar entregas rápidas y continuas de software funcionando.

➔ **Concibe al proyecto en partes homogéneas**

La capacidad de “dividir” al proyecto en partes capaces de adaptarse sobre la marcha, complementarse y resolverse en poco tiempo, ayuda a que si hay que realizar cualquier modificación, sólo se hacen cambios en la parte implicada y en poco tiempo.

➔ **Promueven el trabajo colaborativo**

Además de los beneficios en los procesos, también fomenta el trabajo multidisciplinario, la autonomía y transparencia. Como todas las partes persiguen un objetivo en común y trabajan con fluidez y flexibilidad, permite que los equipos obtengan resultados más efectivos.

➔ **Predice resultados y minimiza los riesgos**

Gracias a las revisiones continuas y la adaptación al cambio, permite obtener una mirada predictiva sobre el resultado minimizando los riesgos de cometer errores inmodificables.

➔ **El cliente es un miembro más del equipo**

Gracias a un vínculo fluido con los clientes y un trabajo multidisciplinario, se consiguen resultados satisfactorios lo que hace que el cliente se convierta en un miembro más del equipo, causando proyectos eficientes y una gran experiencia en ellos.

Las metodologías ágiles más utilizadas

Kanban

Existen plataformas online que traducen muy bien esta metodología, tales como: Monday o Trello y que ayudan a entender su particularidad.

Kanban es una palabra japonesa que traducida al español significa “tarjeta visual”. Esta metodología propone una comunicación en tiempo real, controlando el trabajo a través de una línea de producción, en la cual se crean tres columnas: pendientes, en proceso y terminadas. Esto permite clasificar las tareas y visualizar fácilmente el avance de las mismas.

Extreme Programming o XP

El punto clave de esta metodología es que fue creada para responder a ambientes muy cambiantes donde se necesita una retroalimentación permanente. Busca poner más

énfasis en la adaptabilidad de un proyecto que en su previsibilidad, ya que asegura que así se conseguirá el resultado esperado.

En consecuencia, quienes participan en esta metodología entienden que los cambios son inevitables y, de hecho, más beneficiosos que un crecimiento estático.

Scrum

Su gran característica es que se lleva adelante en “Sprints”, es decir, procesos de trabajo que deben ser lo más cortos posibles. Al finalizar cada sprint, el equipo debe entregar una versión mejorada del proyecto para que sea analizada por el Owner y los demás interesados, los cuales darán una devolución, para luego iniciar con el proceso de mejora.

Propone una forma de trabajar donde se presentan diversos roles, tales como:

- **Scrum Master:** facilita la aplicación del método de trabajo y gestiona cualquier cambio necesario.
- **Product Owner:** representa a los stakeholders (clientes u otras figuras interesadas en que el proyecto salga correctamente)
- **Stakeholder:** Es el cliente, el cual debe definir los requerimientos y proporcionar el feedback.
- **Team:** las personas que ejecutan o producen el producto.

Scrum es un marco de trabajo en el que equipos cross-funcionales pueden crear productos o desarrollar proyectos de una forma iterativa e incremental. El desarrollo se estructura en ciclos de trabajo llamados Sprints (también conocidos como iteraciones). Estas iteraciones no deben durar más de cuatro semanas cada una (siendo dos semanas la duración más habitual) y tienen lugar una tras otra sin pausa entre ellas.

Los Sprints están acotados en el tiempo– finalizan en una fecha determinada independientemente de si el trabajo ha finalizado por completo o no, y jamás se prorrogan. Normalmente los equipos Scrum escogen una duración de Sprint y la mantienen para todos sus Sprints hasta que mejoran y pueden emplear ciclos más cortos. Al principio de cada Sprint, un Equipo cross-funcional (de en torno a siete personas) selecciona elementos (peticiones del cliente) de una lista priorizada. El equipo acuerda un objetivo colectivo respecto a lo que creen que podrán entregar al final del Sprint, algo que sea tangible y que estará “terminado” por completo.Durante el Sprint no se podrán añadir nuevos elementos; Scrum se adapta a los cambios en el

siguiente Sprint, pero el pequeño Sprint actual está pensado para concentrarnos en un objetivo pequeño, claro y relativamente estable.

Todos los días el Equipo se reúne brevemente para inspeccionar su progreso y ajustar los siguientes pasos necesarios para completar el trabajo pendiente. Al final del Sprint, el Equipo revisa el Sprint con los diferentes Stakeholders (interesados e involucrados en el producto) y realiza una demostración de lo que han desarrollado. Se obtiene feedback que podrá ser incorporado en el siguiente Sprint. Scrum enfatiza un producto "funcionando" al final del Sprint que esté realmente "terminado".

En el caso del software, esto significa un sistema que está integrado, testado, con la documentación de usuario generada y potencialmente entregable.

BIBLIOGRAFÍA

[1] Python para todos. R. Gonzalez Duque. Licencia Creative Commons Reconocimiento 2.5 España.
Descarga gratuita de versión más reciente en http://mundogeek.net/tutorial-python/

[2] Python 3 al descubierto 2da Edición. A. Fernández Montoro. Alfaomega. Año 2013

[3] Tutoriales Web varios.