



# Prueba - Catalog Development - 2021

Objetivo:

Demostrar las capacidades técnicas tanto teóricas como prácticas, manejo de consultas SQL, Python/Flask y GIT

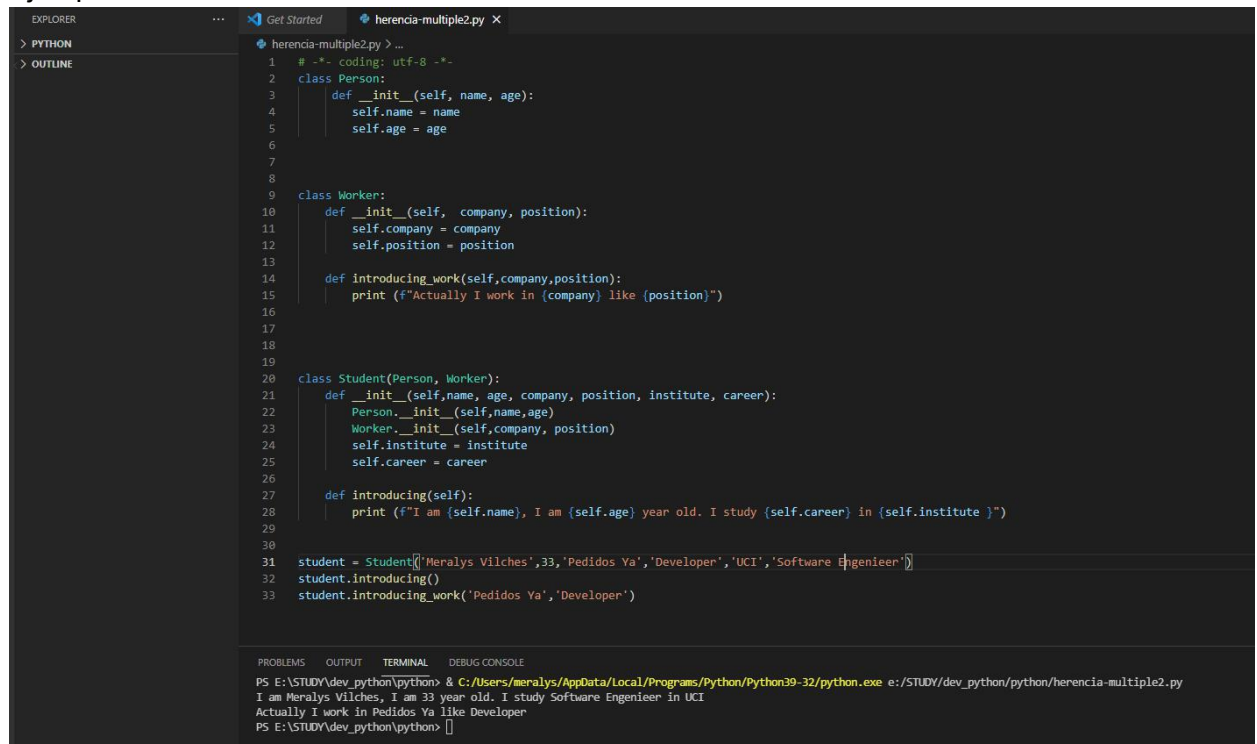
La prueba consta de 4 secciones:

- Teorico
  - Preguntas
- Practica
  - SQL
  - Python-Flask
  - Frontend
  - GIT

## Teorico - Preguntas

- ¿Qué es una clase? ¿Es una clase igual a un objeto? Justifique.  
¿Qué es una clase? Una clase es una plantilla en la cual se representan atributos(propiedades de la clase) y métodos (todo lo que opera sobre los atributos de la clase).  
¿Es una clase igual a un objeto?. No es una clase igual a un objeto, la clase es la plantilla en la cual podemos representar lo anteriormente explicado y el objeto es la instancia (referencia) de una clase, una clase puede tener uno o varios objetos diferentes, cada objeto que se crea representa una instancia de una clase.
- ¿Existe diferencia entre una clase abstracta y una interfaz? Justifique.  
Una interfaz contiene métodos definidos sin código, mientras que una clase abstracta contiene también un conjunto de métodos, pero no es necesario que todos los métodos esten vacíos, algunos pueden tener código.
- ¿Existe la herencia múltiple? Dar un ejemplo  
En python si existe la herencia multiple, a diferencia de la herencia simple vamos a tener una clase que hereda de varias clases padres

### Ejemplo



```
EXPLORER
> PYTHON
> OUTLINE
herencia-multiple2.py
1  # -*- coding: utf-8 -*-
2  class Person:
3      def __init__(self, name, age):
4          self.name = name
5          self.age = age
6
7
8
9  class Worker:
10     def __init__(self, company, position):
11         self.company = company
12         self.position = position
13
14     def introducing_work(self, company, position):
15         print (f"Actually I work in {company} like {position}")
16
17
18
19
20 class Student(Person, Worker):
21     def __init__(self, name, age, company, position, institute, career):
22         Person.__init__(self, name, age)
23         Worker.__init__(self, company, position)
24         self.institute = institute
25         self.career = career
26
27     def introducing(self):
28         print (f"I am {self.name}, I am {self.age} year old. I study {self.career} in {self.institute}")
29
30
31 student = Student('Meralys Vilches', 33, 'Pedidos Ya', 'Developer', 'UCI', 'Software Engineer')
32 student.introducing()
33 student.introducing_work('Pedidos Ya', 'Developer')
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS E:\STUDY\dev_python\python> & C:\Users\meralys\AppData\Local\Programs\Python\Python39-32\python.exe e:/STUDY/dev_python/python/herencia-multiple2.py
I am Meralys Vilches, I am 33 year old. I study Software Engineer in UCI
Actually I work in Pedidos Ya like Developer
PS E:\STUDY\dev_python\python>
```

- ¿Que diferencia existe entre las variables const, let y var en Javascript?  
Tanto var como const y let son usados para la declaración de variable, cons y let son introducidos a partir de ES6 con el objetivo de hacer que nuestro código javascript sea menos propenso a error en comparación con el uso de var.

Las variables declaradas con var son **globalmente scoped** o scoped localmente a una función además de ser **hoisted**. Eso significa que cualquier variable declarada con var fuera de una función pertenecen al objeto global windows además de ser posible su re declaración y actualización lo que tiende en muchos caso a ser una de las primeras razones de errores en nuestro código. El hosting es un mecanismo de javascript que mueve hacia el scope global la declaración de funciones y variables, en el caso de las variables antes de su ejecución son inicializadas con undefined.

Las variables declaradas con **let** son block scoped es decir tienen ámbito en el bloque de código que son declaradas, las variables con let permiten ser actualizadas pero no re declaradas, **let** tambien es hoisted pero a diferencia de **var** no son inicializada por tanto si tratamos de usarla antes de su declaración podemos apreciar un error de referencia.

Las variables o constantes declaradas con **const** son muy similares a **let**, son block scoped pero una vez inicializadas no permiten modificación, por eso deben ser inicializadas en el momento de su declaración, las declaraciones con **const** también son **hoisted** pero en su ámbito local como **let**.

- ¿Existe alguna diferencia entre DOM y Html? Justifique.  
Aunque pueda parecer lo mismo no lo es, HTML(Hypertext Markup Language) es un conjunto de etiquetas que se unen para formar un lenguaje que pueda ser parseado e interpretado por el navegador convirtiéndolo en el DOM(Document Object Model) permitiendo así el uso de javascript para dinámicamente poder adicionar eliminar modificar elementos haciendo uso de la API del DOM. El html pasa de ser un simple texto por asi decirlo a un objeto que puede ser manipulado con javascript.
- ¿Cuál es la diferencia entre asincronía y paralelismo? ¿Son conceptos excluyentes?  
La **asincronía** se pone de manifiesto al hacer la llamada a un recurso x del cual debes esperar una respuesta pero no nos interesa bloquear el hilo principal de nuestro programa por lo que continúa la ejecución una vez resuelta la petición existen mecanismo para volver al punto donde fue llamado y completar las instrucciones asociadas a esa llamada.  
El caso del **paralelismo** es la capacidad de poder realizar peticiones a recursos de forma independientes o ejecutar instrucciones de forma independiente



dígase ejecutarse en hilos independientes pudiendo ser o no concurrentes. Son conceptos excluyentes la asincronía se pone de manifiesto en operaciones de entrada y salida mientras que el paralelismo está más enfocado al uso de la cpu.

- El paralelismo y la concurrencia. ¿Son lo mismo?  
Están relacionados pero no son lo mismo, la concurrencia es la capacidad del CPU para ejecutar más de un proceso al mismo tiempo. Puede decirse que es la habilidad que posee ciertas partes de un programa, algoritmo, o problema la cual puede ser ejecutado en orden parcial, sin afectar el resultado final. Los cálculos (operaciones) pueden ser ejecutados en múltiples procesadores. En caso de que la computadora solo tenga una CPU o un núcleo, es posible que la aplicación no avance en más de una tarea al mismo tiempo. En su lugar, divide el tiempo y el núcleo / CPU entre varias tareas. Por otra parte el paralelismo dado un problema inicial divide el problema en fracciones más pequeñas, para luego procesarlos de forma concurrente, aprovechando al máximo la capacidad del procesador para resolver el problema. La aplicación debe tener más de un hilo ejecutándose en núcleos separados. En una CPU de un solo núcleo, el paralelismo no es posible.

- ¿Qué es `__init__` en Python?  
El `__init__` en python es un método constructor que es el encargado de realizar las tareas de inicialización de los atributos del objeto que se crea, se encarga de construir los objetos con sus características, se ejecuta cuando se crea un nuevo objeto o cuando se crea una nueva instancia de la clase.

- ¿Puede explicar `*args` y `**kwargs` en Python?

**\*args**, es una lista de argumentos que se usa cuando no se está seguro de cuantos argumentos se tengan que pasar a la función; es decir permite pasar una cantidad "x" de argumentos.

**\*\*kwargs**: es un diccionario, se usa cuando no se sabe el número de argumentos que va a recibir una función

La sintaxis es `*` y `**`, es decir que si a una función le pasamos como argumento `*lista` o `**diccionario` funciona de igual manera

- ¿Que es un decorador en Python?  
Los decoradores son funciones que modifican el comportamiento de otras funciones, ayudan a acortar el código. Los decoradores pueden ser útiles ya que

permiten la extensión de una función existente no realizando ninguna modificación sobre la función original

- ¿Cuáles de las siguientes frases crean un diccionario en el lenguaje Python?

(Múltiples respuestas correctas posibles)

- a) `d = {}`
  - b) `d = {"juan":40, "pedro":45}`
  - c) `d = {40: "juan", 45: "pedro"}`
  - d) `d = (40: "juan", 45: "50")`
- Respuesta: c) `d = {40: "juan", 45: "pedro"}`

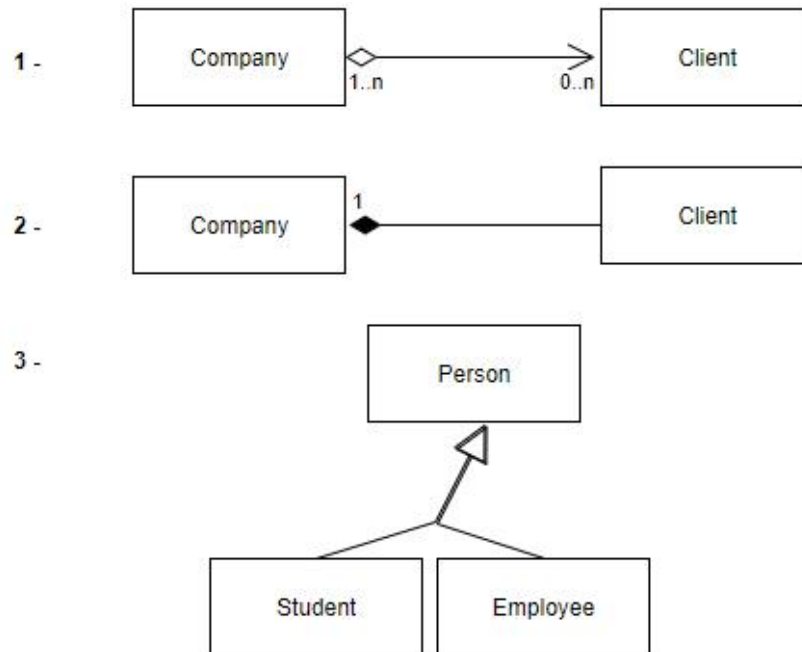
- ¿Qué imprime este código?

```
try:
    if '1' != 1:
        raise "algún error"
    else:
        print("no se ha producido algún error")
except "algún error":
    print("se ha producido algún error")
```

- a) se ha producido algún error
- b) no se ha producido algún error
- c) código inválido
- d) ninguno de los anteriores

Respuesta: d) ninguno de los anteriores

- Dado los siguientes diagramas UML. Indique cual es correspondiente a composición, cual a herencia y cual define una agregación.



1- Es una relación de agregación

2- Es una relación de composición

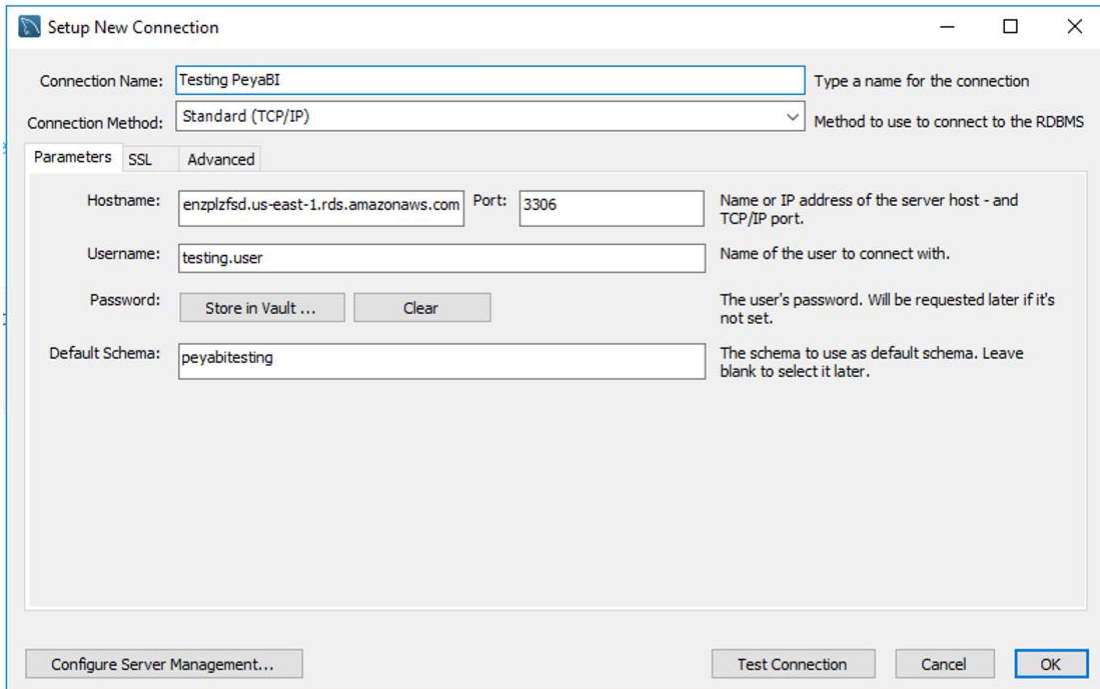
3- Define una herencia

# Practica - Prueba SQL

## Preparación de ambiente

Prepara el ambiente para la prueba de SQL

1. Descargar Mysql Workbench para la conexión a la base de datos de Testing MySQL. <https://www.mysql.com/products/workbench/>
2. Instalar, configuraciones por defecto.
3. Crear una nueva conexión con los datos:
4. hostname: peyabi-testing.crdenzplzfsd.us-east-1.rds.amazonaws.com  
port: 3306  
username: testing.user  
password: m129V9x8n3187yCN7Q1C  
Default Schema: peyabittesting



Setup New Connection

Connection Name:  Type a name for the connection

Connection Method:  Method to use to connect to the RDBMS

Parameters SSL Advanced

Hostname:  Port:  Name or IP address of the server host - and TCP/IP port.

Username:  Name of the user to connect with.

Password:   The user's password. Will be requested later if it's not set.

Default Schema:  The schema to use as default schema. Leave blank to select it later.



## Consultas

Crear consultas SQL que respondan a las siguientes solicitudes.

1. Cantidad de órdenes y monto total de órdenes por ciudad, país. Nivel año-mes
2. Cantidad de restaurantes en Uruguay, por categoría de restaurante, que tienen por lo menos un monto mensual (order amount) en abril 2017 mayor a 1000.
3. Cantidad de restaurantes distintos en que compró cada usuario en todo el período.

El resultado de esta prueba debe ser un archivo .sql que contenga todas las consultas. En caso que considere pertinente, por favor adjuntar comentarios en el script de forma de facilitar la lectura.





## Prueba Python-Flask

En esta sección será necesario desarrollar una API-REST en Flask la cual deberá exponer 4 endpoints.

Para fuente de datos, se brindan tres archivos JSON los cuales deberán ser levantados en memoria a modo de simular una base de datos.

Archivos:

1. El archivo **comments.json** contiene todos los comentarios disponibles.
2. El archivo **users.json** contiene todos los usuarios disponibles.
3. El archivo **albums.json** contiene todos los álbumes disponibles.

Se pide los siguientes endpoints:

- Todos los comentarios disponibles
- Dado un id de posteo, retornar todos los comentarios relacionados a dicho posteo.
- Retornar el posteo con mayor cantidad de comentarios asociados. Si existe más de uno, devolverlos todos.
- Dado un id de usuario, devolver los álbumes que tiene asociado.

Nota:

Se valorará la prolijidad del código escrito.

En caso que considere pertinente, por favor adjuntar comentarios en el script de forma de facilitar la lectura.

## Frontend

La idea es convertir el siguiente CV, en una ONE PAGE utilizando: HTML, CSS y JS.  
La página debe ser responsiva, se puede utilizar frameworks/librerías de JS y/o CSS.

---

### Curriculum Vitae



Nombre: Juan  
Apellido: Perez  
Nacionalidad: Uruguay  
Fecha de Nacimiento: 01/01/1996

Educación: Carrera X en Universidad Y

Experiencia laboral:

- 2018-2020 - Empresa 1
- 2020-Actualidad - Empresa 2

Cursos:

- Curso 1 - Academia 1
- Curso 2 - Academia 1
- Curso 3 - Academia 1

Idiomas:

- Ingles
  - Español
-

## Práctica - conocimientos de GIT

En esta sección se evalúa el conocimiento de la herramienta de versionado GIT.  
Lea atentamente los pasos y recuerde respetar las nomenclaturas correspondientes.

- Crear un repositorio **público** en su cuenta de Github con el nombre **peya-prueba-{nro\_cedula}**.  
En caso de no contar con una, puede crearla gratis desde el siguiente [enlace](#).
- Deberá crear la rama **develop** dependiente de master y realizar el push al repositorio.
- Desde **develop**, deberá crear la rama **ft\_prueba\_teorica** y subir el documento de respuestas de las preguntas teóricas en formato pdf.
- Desde **develop**, deberá crear la rama **ft\_prueba\_python\_flask** y subir el código fuente realizado para resolver la prueba python-flask.
  - Nota: El código será ejecutado por el equipo corrector de la prueba. Es por esto que pedimos en caso de requerir algún paso previo a su ejecución, por favor adjuntar nota al inicio del script con pasos.
  - Nota: Se evaluarán buenas prácticas de programación.
- Desde **develop**, deberá crear la rama **ft\_prueba\_sql** y subir las queries correspondientes la prueba sql. Se deberá enviar las consultas en formato .sql
- Desde **develop**, deberá crear la rama **ft\_prueba\_frontend** y subir la pagina de la prueba de frontend.
- Se deberá hacer un merge de las ramas abiertas (**ft\_prueba\_teorica**, **ft\_prueba\_python\_flask**, **ft\_prueba\_sql** y **ft\_prueba\_frontend**) a **develop**.
- Se deberá hacer un merge de **develop** con la rama **master**.

**Nota: Es de suma importancia que se adjunte al momento de entregar la prueba el link hacia el repositorio creado.**