**ABPro 4,5,6**

**\*Cursiva=** documentación acerca de preguntas/indicaciones de cada pdf.

**ABPro4:**

1. Agregar una clase **BodegaPrincipal:**
   * atributos:
     1. Dirección
     2. Mts2 ( metros cuadrados )
     3. Stock
   * funciones:
     1. **despachar\_producto**: **descontará** un valor desde su **stock** y luego **sumará** a una **sucursal** (aumentando su cantidad de stock).
     2. **recepcionar\_producto:** sumará valor al stock.
2. Agregar clase **Sucursal:**
   * hereda de clase **BodegaPrincipal:**
     1. + método **despachar\_producto: descontar stock** de la sucursal y **agregarlo** al **stock** de la clase **BodegaPrincipal**.
3. En clase producto:
   * atributo stock: valor→**Composición** de clase Sucursal
     1. contará con **límite** de stock de **500** productos,
        + agrega o sobrepasa límite→
          - emitir **mensaje** de despacho del producto
          - **sumarlo** directamente en el atributo **Stock** de **BodegaPrincipal**.
4. **Identifique** casos en que **polimorfismo** solucionaría problemas de herencia de métodos y atributos.
5. **Definan** la **utilidad** de MRO para determinados ejercicios de herencia.
   * ***Orden de resolución de método*** *(MRO) indica la forma en que un lenguaje de programación resuelve un método o atributo.En Python, el orden de resolución del método define el orden en el que se buscan las clases base al ejecutar un método. Primero, se busca el método o atributo dentro de una clase y luego sigue el orden que especificamos al heredar. Este orden también se denomina* ***Linealización*** *de una clase y el conjunto de reglas se denomina* ***MRO****(Orden de resolución de método).*

*Mientras hereda de otra clase, el intérprete necesita una forma de resolver los métodos que se llaman a través de una instancia. Por tanto, necesitamos el orden de resolución de método.*

*New style classes(Py 3.x & 2.2) →* ***C3 Linearization algorithm***

1. **Entrega**: archivo py

**ABPro5:**

1. Si atributo **stock** de la clase **Sucursal** cuenta con un stock **menos de 50:**
   * automáticamente mostrar mensaje indicando que se está solicitando y reponiendo productos.
   * clase **Bodega** se deberá **descontar 300** del atributo **stock** y **sumarlos** a **Sucursal**.
   * **Insuficiente** stock en la clase Bodega:
     + **Mensaje** → no existe stock suficiente para reponer.
2. implementar la función **super()** para acceder a atributos y métodos de clases superiores.
   * ***Using Super():*** *Python super() function provides us the facility to refer to the parent class* ***explicitly****. It is basically useful where we have to* ***call superclass functions****. It returns the proxy object that allows us to refer parent class by ‘super’.*
3. Implementar clase **OrdenCompra** :
   * Atributos:
     + **Id\_ordencompra**
     + **producto**:
       - **composición** de la clase Producto y el atributo **despacho**,
       - solo almacenará valores booleanos.
     + **despacho:** sea True ( Verdadero )
       - → **agregar** al valor del producto **5.000 CLP** por despacho y
4. función vender de la clase **Vendedor:** mostrar por consola **total final** con el **detalle**

( valor neto, impuesto, despacho, valor total )

1. **Entrega**: archivo py

**ABPro6:**

DESARROLLO

1. Incluir las sentencias **try-except** → capturar errores en nuestro sistema. → Done!
   * valor incorrecto:
     + no se termine programa
     + muestre un mensaje de error correspondiente.
2. Crear una Excepción personalizada (Excepción definida por el usuario), la cual nos permite realizar una acción al momento de que no exista stock de productos en la clase **Producto**, **Sucursal** y **Bodega**.
   * Clase de Excepción:
     + en un archivo externo e importar al proyecto principal.
3. Carro de compras:
   * genere error en caso que cliente quiera comprar **más de 10** unid. de un producto.
   * Cree un método para mostrar el **valor promedio** de las **compras** de un cliente.
     + Realice el trabajo necesario para tratar el error en caso que el cliente tenga solo 0 compras.
4. Cada vez que se trabaje un error se debe retornar True.
5. **Entrega**: archivo py

**ABPro6**:

class Excepcion:

#Chequear si stock de producto1= Producto(), sucursal1=Sucursal() y bodega1=Bodega() > 0

def sin\_stock( x ):

try:

x.stock >= 1

except:

print(f'{x.nombre} no tiene stock')

else:

print(f'El stock actual es de {x.stock } unidades.')

#Chequear que carro de clientes tenga máx 10 productos. De lo contrario, enviar mensaje de error.

def max\_compra ( cliente):

len(cliente.carrito)<= 10

try :

print(f'Su carro contiene {cliente.carrito} unidades.')

except:

delete= len(cliente.carrito)-10

if delete >=1 :

print(f' Debe eliminar {delete} items de su carrito')

print(f' ¿Qué desea eliminar?')

print (cliente.carrito)

eliminar= input('Producto a eliminar:')

if eliminar in cliente.carrito:

cliente.carrito.remove(eliminar)

finally:

print(f' {cliente.id\_cliente},gracias por comprar en nuestra tienda.')

#Obtener valor promedio de compras del cliente y enviar mensaje en caso de que sea 0.

def compra\_promedio (cliente, compras ):

try :

resultado= sum(cliente.compras)/ 2

except:

print(f' {cliente.id\_cliente} no ha realizado compras')

else:

print(f'La compra promedio de {cliente.id\_cliente} equivale a {resultado}')

#(\*Si “compras” es una lista en el constructor de esa clase: sum(self.compras))

**ABP**

**1,2,3**

1. 3 usuarios de su aplicación:
   * 3 atributos por c/u.
   * 3 acciones por c/u Las acciones se deben ejecutar de forma interna en nuestra nuevos atributos y acciones que pueden realizar los objetos
2. nueva clase
   * 4 atributos (uno de ellos debe ser opcional).
   * Crear métodos para c /usuario.
3. Diferentes acciones particulares p/clase.
4. 4 métodos p/clase.
   * 2→ afecten números
   * 2→ que afecten strings.
   * ‘sobrecarga de métodos’.
5. existan condiciones para realizar las validaciones correspondientes.
6. Incluyendo: - Generalización. - Realización. - Dependencia - Agregación - Asociación unidireccional o bidireccional, dependiendo del problema. - Composición.

**4,5,6**

En base al diagrama de clases del ejercicio anterior

1. Integra una estructura de herencia de **tres niveles.**
2. Genere una clase **principal**, para luego desarrollar **perfiles** más **particulares**.
3. Aplique e integre el orden de resolución de métodos.
   * C3 Linearization Algorithm works on three rules:

* Inheritance graph determines the structure of method resolution order.
* User have to visit the super class only after the method of the local classes are visited.
* Monotonicity
* To get the MOR: **\_\_mro\_\_** attribute or **mro()** method # display the order in which methods are resolved. For Example:
  + print((class name).\_\_mro\_\_) → User.\_\_mro\_\_
  + print((class name).mro())

# invoking the \_\_init\_\_ of the parent class → (dentro del child init)

Person.\_\_init\_\_(self, name, idnumber)

1. Qué utilidad tiene el **polimorfismo** en la definición/creación de métodos heredados. I
2. Indique en qué momento es **conveniente** utilizar las funciones **isinstance()** y **issubclass(**).
3. **Entrega**: archivo py
4. Agregue un **método por** cada **clase** creada.
5. Realice ejercicios para comprobar herencia de métodos y atributos.
6. Incorpore un ejemplo práctico de **sobreescritura** de métodos en su ejercicio individual.
   * *overriding method.*
7. **Entrega**: archivo py
8. Identifica al menos **seis posibles errores** en la ejecución de su programa.
   * SyntaxError: invalid syntax:
     1. Syntax errors, aka parsing errors. Most common in the learning process.
     2. When the proper syntax of the language is not followed then a syntax error is thrown.
     3. The error is caused by (or at least detected at) the token *preceding* an arrow.
   * Exception: logical errors.
     1. occurs after passing the syntax test
        1. **IOError:** if the file can’t be opened
        2. **KeyboardInterrupt:** when an unrequired key is pressed by the user
        3. **ValueError:** when built-in function receives a wrong argument
        4. **EOFError:** if End-Of-File is hit without reading any data
        5. **ImportError:** if it is unable to find the module (not found)
        6. **ZeroDivisionError**: division by zero
        7. **IndentationError**: expected an indented block
        8. **IndexError**: When the wrong index of a list is retrieved.
        9. **AssertionError**: It occurs when the assert statement fails
        10. **AttributeError**: It occurs when an attribute assignment is failed.
        11. **KeyError**: It occurs when the key of the dictionary is not found.
        12. **NameError**: It occurs when the variable is not defined.
        13. **MemoryError**: It occurs when a program runs out of memory.
        14. **TypeError**: It occurs when a function and operation are applied in an incorrect type.
9. Integre por lo menos **IndexError**, **TypeError**, **KeyError**.
10. Ejecuta un **Finally** al final de **cada** manejo de **error**:
    * entregar mensaje que **explique** error y **forma** de **solucionarlo**. (No a lugar. 50% ineficiencia)
11. **Describa** cual es la **utilidad** de **manejar** los **errores** en la programación.
    * *Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occur which changes the normal flow of the program.*
    * *When an error and an exception are raised then we handle that with the help of the Handling method.*
    * *Exception handling in Python not only provides a way to respond to errors, it* ***allows you to elegantly structure your code and solve problems, reduces code, and sometimes speeds up your script****.*
    * *We can handle errors by the Try/Except/Finally method. We write unsafe code in the try, fall back code in except and final code in finally block.*
12. **Envíe** el **diagrama** de Clases desde Object hasta los tipos de errores, tal como se muestra en los videos.
13. **Describa** la estructura del diagrama de forma **detallada**.
14. **Entrega**: Script .PY

**Try-Except:**

* + The try block is used to check some code for errors
  + A [try](https://docs.python.org/3/reference/compound_stmts.html#try) statement may have **more than one** *except clause*, to specify handlers for different exceptions.
  + If the exception is left unhandled, then the execution stops.