# Examen Odoo - HNET


### Edgar Josué Benedetto Godoy

### 0801-1997-23600

**ejbg597@gmail.com**

**edgar.benedetto@unah.hn**

**+504 3330-0171**


**01/07/2021**

---

---

### Consideraciones

- Corroborar que el **path** de los ***addon custom*** se encuentre definido en el archivo de configuración de Odoo en la sección de **addons_path**. Por ejemplo si la carpeta donde se incluiran los modulos customizados tiene por nombre **custom_addons** y se encuentra en la ruta: "**_opt/odoo13/custom*addons***", dicha ruta debe ser agregada en el campo **addons_path** quedando de esta forma:
  - **_addons_path= opt/odoo13/odoo/addons, opt/odoo13/addons, opt/odoo13/custom*addons***
- Ejemplo Propuesto:
  - Una empresa que tiene como principal product la venta de bebidas, carácterizandose por vender batidos nutricionales y bebidas alcoholicas de alta calidad necesita poder diferenciar de forma efectiva cuando **(2,3) un producto contienen o no contienen alcohol**, también se necesita **(5)calcular el impuesto** de los productos que contengan alcohol

---

---

## Pregunta #1

---

### ¿Qué componentes básicos debe de llevar un addon para que funcione en Odoo?

#### Componentes básicos de un addon para que funcione en Odoo

1. Archivo Manifiesto
   - Es un archivo en Python que sirve para especificar los metadatos del módulo contenidos en un diccionario y permite visualizar el módulo dentro de la lista de aplicaciones de Odoo. Se le da el nombre de `__manifest__.py` .

- Ejemplo:
  - Nombre del módulo: **sales_extension**
  - Nombre del archivo:

    `__manifest__.py`
  - Contenido del archivo:

In [ ]:

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------

    @author edgar.benedetto@unah.hn
    @date 01/07/2021
    @decription Archivo manifiesto del módulo extensión de ventas
    @name_file __manifest__.py
    @version 1.0

-----------------------------------------------------------------------------------------------------
    Archivo Manifiesto - Extensión del módulo de Ventas
-----------------------------------------------------------------------------------------------------
"""
{
  'name': 'Sales Extension', #Nombre del modulo
  'version': '1.0', #Versión en la que se encuentra el modulo
  'category': 'Sales/Sales', #Categoría de clasificación o dominio empresarial dentro de Odoo
  'author': 'Edgar Benedetto', #Autor del Moulo
  'depends': ['sale'], #Modulos que deben cargar antes del actual
  'data': [
```

```
        'views/product_extension.xml'
    ], #Lista de rutas de archivos de datos que siempre deben instalarse o actualizarse con el módulo.
    'installable': True, #Modulo instalable
}
```

1. Archivo descriptor
   - Sirve para cargar paquetes de Python en Odoo y también proporciona una descripción del módulo, tiene como carácteristica que se ejecuta desde el inicio del programa. El archivo descriptor lleva el nombre de `__init__.py` .

- Ejemplo:
  - Nombre del módulo: **sales_extension**
  - Nombre del archivo:

    `__init__.py`
  - Contenido del archivo:

In [ ]:

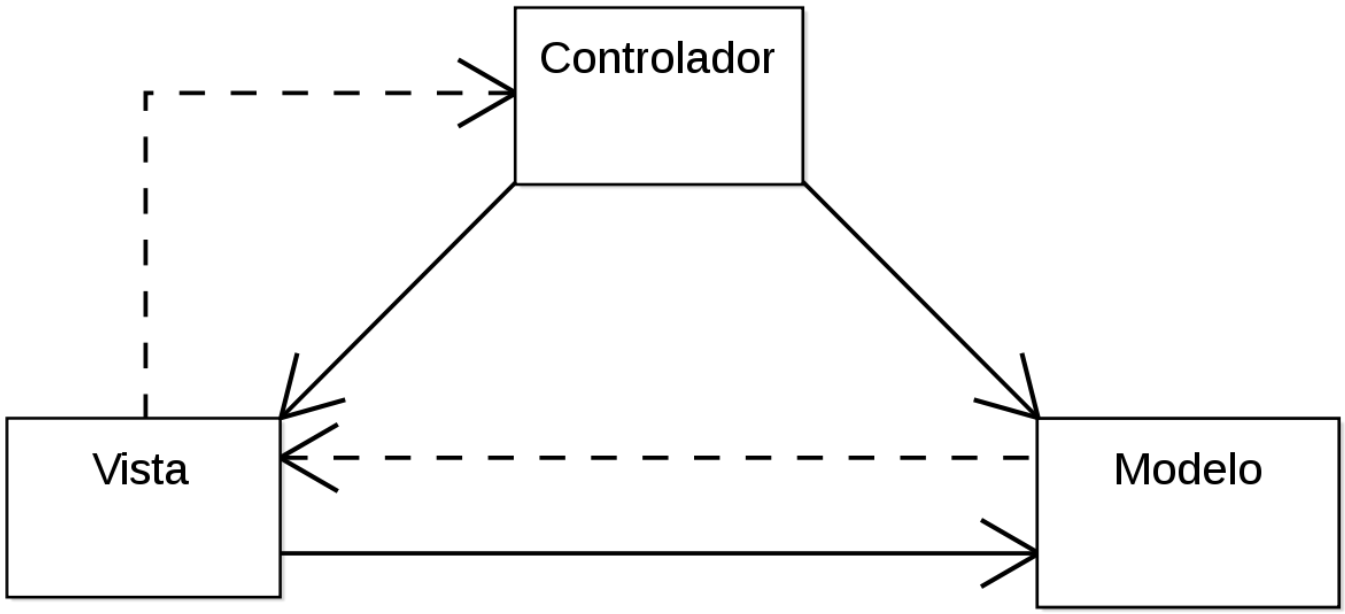
```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------------

    @author edgar.benedetto@unah.hn
    @date 01/07/2021
    @decription Archivo descriptor del módulo extensión de ventas
    @name_file __init__.py
    @version 1.0

-----------------------------------------------------------------------------------------------------------
    Archivo Descriptor - Extensión del módulo de Ventas
-----------------------------------------------------------------------------------------------------------
"""

# Importar Modelo-Vista-Controlador
import . from models
import . from views
# import . from controllers
```

1. Modelo-Vista-Controlador




- En Odoo se hace uso del estilo de arquitectura de software que separa los datos de un módulo en tres componentes distintos.
  1. El Modelo que contiene la lógica de negocio y sus mecanismos de persistencia.
  2. La Vista o interfaz de usuario, es la capa de aplicación de los datos y se compone de la información que se envía al cliente junto a las formas de interactuar con dicha interfaz.
  3. El Controlador actúa de intermediario entre el Modelo y la Vista gestionando el flujo de información y transformación de los datos entre ellos.

---

---

## Pregunta #2

---

### ¿Como crear un modelo en Odoo?

1. Dentro de la carpeta del módulo al cual se le quiere crear un modelo, se debe crear una carpeta comunmente llamada**models**
2. Dentro de la carpeta **models** se debe crear un archivo descriptor el cual lleva el nombre: `__init__.py` .

- Ejemplo:
  - Nombre del módulo: **sales_extension**
  - Nombre del modelo: **alcoholic_drink**
  - Nombre del archivo:

    `__init__.py`
  - Contenido del archivo:

In [ ]:

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------

    @author edgar.benedetto@unah.hn
    @date 01/07/2021
    @decription Archivo descriptor del modelo del módulo extensión de ventas
    @name_file __init__.py
    @version 1.0

-----------------------------------------------------------------------------------------------------
    Archivo Descriptor del Modelo - Extensión del módulo de Ventas
-----------------------------------------------------------------------------------------------------
"""

# Importar los modelos creados
import . from alcoholic_drink
import . from isv_alcoholic_drink
import . from get_jun_sales_orders
```

1. Crear los archivos en Python con nombres carácteristicos para cada uno de los modelos necesarios.
   - En el modelo se debe escribir el atributo más importante el cual es **_name** es obligatorio y define el nombre del modelo en el sistema Odoo.
   - Se puede heredar o usar cualquier módulo | objeto | clase | modelo | vista | un solo campo de módulos existentes. Algunos de los usos de la herencia son:
     A. Cambiar los atributos de algunos campos existentes.
     B. Agregar | modificar | eliminar campos antiguos o nuevos en el modelo existente | personalizado
     C. Agregar botones en la vista de modelo ya existente | personalizado

- Ejemplo:
  - Nombre del módulo: **sales_extension**
  - Nombre del modelo: **alcoholic_drink**
  - Nombre del archivo:

    ```
    alcoholic_drink.py
    ```
  - Contenido del archivo:

In [ ]:

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------

    @author edgar.benedetto@unah.hn
    @date 01/07/2021
    @decription Modelo para agregar un campo que permite verificar si un producto en venta es
      una bebida alcoholica o no
    @name_file alcoholic_drink.py
    @version 1.0

-----------------------------------------------------------------------------------------------------
    Modelo Bebida Alcoholica - Extensión del Producto en venta
-----------------------------------------------------------------------------------------------------
"""

# Importar de odoo modelos y campos
import odoo from models, fields

class AlcoholicDrink(models.Model):
  """
  Crear campo para poder determinar cuando un producto contiene alcohol o no
  """
  # Nombre del modelo
  _name = 'alcoholic_drink'

  # Heredar el template de producto
  _inherit = 'product.template'

  # Campo a agregar en la vista para determinar si un producto es una bebida alcoholica
  _is_alcohol = fields.Boolean(string='Alcoholic Drink')
```

## Pregunta #3

### ¿Cuál es la sintaxis básica para heredar a una vista en Odoo?

- La característica que permite interactuar con los objetos subyacentes sin modificar directamente el objeto original es la **herencia**, funciona como un agregado de capas para la modificación por encima de los objetos existentes.
- Las especificaciones de herencia se componen de un**localizador de elementos**, para que coincida con el elemento heredado en la vista principal, y el elemento secundario que se utilizará para modificar el elemento heredado. Hay tres tipos de localizadores de elementos para hacer coincidir un elemento de destino pero para heredar una vista se hace uso solamente de uno:
  1. Un elemento **campo | field** con un atributo **nombre | name**, la coincidencia se obtiene con el primero campo que tenga el mismo nombre, ignorando todos los demás atributos durante la comparación.
  2. Sintaxis básica para heredar una vista en Odoo
     ```
     <field name="inherit_id" ref="library.view_book_form"/>
     ```
- Ejemplo:
  - Nombre del módulo: **sales_extension**
  - Nombre del modelo: **alcoholic_drink**
  - Nombre del archivo:

    ```
    alcoholic_drink.xml
    ```
  - Contenido del archivo:

    ```
    <?xml version="1.0" encoding="UTF-8" ?>
    <!--
        @author edgar.benedetto@unah.hn
        @date 01/07/2021
        @decription Vista para agregar el campo de verificación para un
          producto en venta que contiene alcohol
        @name_file alcoholic_drink.xml
        @version 1.0

        ============================================================================
        Vista Producto - Extensión del Producto en venta para agregar el campo
          de verificación para un producto en ventaque contiene alcohol
        ============================================================================
    -->
    <odoo>
      <data>
          <record id="product_template_only_form_view_id" model="ir.ui.view">
              <field name="name">product.template.product.form.od</field>
              <field name="model">product.template</field>

              <!-- Heredar la vista del producto -->
              <field name="inherit_id" ref="product.product_template_only_form_view"/>
              <field name="arch" type="xml">
                  <field name="default_code" position="after">
                      <field name="_is_alcohol"/>
                  </field>
              </field>
          </record>
      </data>
    </odoo>
    ```

## Pregunta #4

### ¿De qué manera se puede heredar un campo nuevo en un modelo existente dentro de Odoo?

- Para heredar solamente un campo en un modelo existente se hace uso del diccionario **_inherits** el cual sirve para mapear el objeto padre.
- La llave del objeto **_inherits** hace referencia al modelo y los valores hacen referencia a los campos del modelo posicionado en la llave.
- Implementa la herencia basada en la composición, el nuevo modelo expone todos los campos de los modelos heredados pero no almacena ninguno de ellos, los valores en sí permanecen almacenados en el registro vinculado.

- Estructura:

```
_inherits = {
  'a.model': 'a_field_id',
  'b.model': 'b_field_id'
}
```

- Ejemplo:

```
class Users(models.Model):
  _name = "res.users"
  _inherits = {'res.partner': 'partner_id'}
```

## Pregunta #5

### ¿Cuál es la sintaxis para crear un campo calculado?

- Los campos se pueden calcular (en lugar de leer directamente de la base de datos) utilizando el parámetro **compute**.
- Los campos calculados son de solo lectura de forma predeterminada. Si se usan los valores de otros campos, se deben especificar dichos campos usando **depends()**.
- Los campos calculados no se almacenan de forma predeterminada, se calculan y devuelven cuando se solicitan. La configuración **store=True** los almacenará en la base de datos y habilitará automáticamente la búsqueda.

- Ejemplo:
  - Nombre del módulo: **sales_extension**
  - Nombre del modelo: **isv_alcoholic_drink**
  - Nombre del archivo:

    ```
    isv_alcoholic_drink.py
    ```
  - Contenido del archivo:

In [ ]:

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------------

    @author edgar.benedetto@unah.hn
    @date 02/07/2021
    @decription Modelo para calcular el impuesto de un producto que contenga alcohol (18%)
    @name_file product_extension.py
    @version 1.0

-----------------------------------------------------------------------------------------------------------
    Modelo ISV Bebida Alcoholica - Extensión del Producto en venta
-----------------------------------------------------------------------------------------------------------
"""

# Importar de odoo modelos, campos y la api de Odoo
import odoo from models, fields, api

class ISVProductAlcohol(models.Model):
  """
  Campo calculado o computado para obtener el precio total de un producto
    que contiene alcohol por lo tanto lleva un
  """
  _name = 'isv_alcoholic_drink'
```

```
  _inherit = 'sale.order' # Heredar orden de venta de sale.py

  @api.depends('order')
  def _compute_isv_alcohol(self):
    """
    Calcular impuesto de un producto que contiene alcohol
    """
    amount_untaxed = amount_tax = 0.0

    for line in self.order_line:
      amount_untaxed += line.price_subtotal
      amount_tax += (line.price_subtotal * 0.18) # Se obtiene el 18% del subtotal de la orden

      # Actualizar los campos en la orden de venta
    order.update({
      'amount_untaxed': amount_untaxed,
      'amount_tax': amount_tax,
      'amount_total': amount_untaxed + amount_tax,
    })

    _isv_alcoholic = amount_untaxed + amount_tax

    return _isv_alcoholic

  # Campo a agregar en la vista para corroborar el precio total con el isv de una bebida alcoholica
  _isv_alcoholic_drink = fields.Float(string='Total (ISV Included)', compute=_compute_isv_alcohol)
```

---

---

## Pregunta #6

---

### Buscar dentro de la tabla “sale.order” los pedidos de venta que se encuentren entre el 1/06/2021 al 31/06/2021. – Mostrar/escribir función correspondiente.

- En algunas situaciones, donde el rendimiento es primordial, se tienen que ejecutar consultas SQL directamente.
- El entorno (env) contiene un atributo **cr**, es un cursor para la base de datos actual y permite ejecutar SQL directamente.
- Se puede imprimir la base de datos usando: **self.env.cr.dbname**, devuelve el nombre de la base de datos actual.
- Al no especificar que datos se requieren especificamente de la tabla sale.order se obtienen todos los datos de los pedidos de venta.
- Respuesta:
  - Nombre del módulo: **sales_extension**
  - Nombre del modelo: **get_jun_sales_orders**
  - Nombre del archivo:

    ```
    get_jun_sales_orders.py
    ```
  - Contenido del archivo:

In [ ]:

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------

    @author edgar.benedetto@unah.hn
    @date 02/07/2021
    @decription Modelo para obtener los pedidos de venta que se encuentran entre
      01/06/2021 al 31/06/2021
    @name_file get_jun_sales_orders.py
    @version 1.0

-----------------------------------------------------------------------------------------------------
    Modelo Pedidos de Junio - Extensión de Pedidos de venta
-----------------------------------------------------------------------------------------------------
"""

import odoo from models, fields, api

class SalesOrdersJun(models.Model):
  """
```

```
  Obtener las ordenes de venta del mes de Junio
  """
  _name = 'get_jun_sales_orders'
  _inherit = 'sale.order'

  def _get_jun_sales_orders(self):
    """
    Utilizar una consulta SQL para obtener todas las ordenes de venta
      del mes de Junio
    """
    self.env.cr.execute("""
      SELECT *
        FROM sale_order
      WHERE
        order.date_order BETWEEN '2021/06/01' AND '2021/06/31'
      """, )
    _orders_jun = self.env.cr.fetchall()
    return _orders_jun
```

---

---

# Pregunta #7

---

### Se le presenta el siguiente requerimiento: Se necesita saber cuántos productos por categoría asignada se encuentran dentro de un pedido de venta.

1. ¿Qué proceso seguiría para obtener esa información?
    - Un cliente de HNET solicita un **apartado al final** del documento de ***"Factura de cliente"*** en el cual se muestren los **subtotales de cantidades** pedidas para las siguientes **categorías de producto: MAQUILLAJE, REPELENTES, ALCOHOL, ESMALTES Y OTROS**. Por último, el cliente solicita un **campo final** con el **total de todas las cantidades** pedidas y contenidas dentro de la factura, el cual es la ***suma de todos los subtotales anteriores.***
2. ¿Cómo agregarías esta información?
3. Notas:
    - Categoría de producto = display_name
    - Cantidades = product_uom_qty

---

### Pasos a seguir para obtener y mostrar la información solicitada

1. **Crear o usar un módulo** destinado a **extender** la **funcionalidad** del módulo encargado de la **factura** de una orden de venta. Implica crear la carpeta con un nombre característico: **_"pos*extension*"**
2. Dentro de la carpeta del módulo "pos_extension" crear los **archivos manifiesto y descriptor del módulo** incluyendo en el manifiesto en la sección de **data** todas las vistas utilizadas para mostrar la información solicitada
3. Crear el directorio ***"models"*** para contener todos los Modelos necesarios
4. Dentro de la carpeta ***"models"*** crear el **archivo descriptor** en el cual se importarán todos los modelos
5. Crear el archivo que contendrá el **modelo** para realizar los cambios solicitado por el cliente en la factura, designandole un nombre característico: **_"category*total.py*"**
6. Crear el directorio encargado de contener las **vistas** necesarias para mostrar la información solicitada: ***"views"***
7. Crear el archivo con extensión ***".xml"*** para **heredar la vista de la factura** y solamente **agregarle los campos** solicitados, designandole un nombre un nombre característico: **_"category*total.xml*"**

---

### Código

1. Archivo Manifiesto del módulo **pos_extension** `__manifest__.py` .
    - Nombre del módulo: **pos_extension**
    - Nombre del archivo:

      `__manifest__.py`
    - Contenido del archivo:

**In [ ]:**

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------

    @author edgar.benedetto@unah.hn
    @date 02/07/2021
```

```
    @decription Archivo manifiesto del módulo extensión de Punto de venta
    @name_file __manifest__.py
    @version 1.0

-----------------------------------------------------------------------------------------------
    Archivo Manifiesto - Extensión del módulo de Punto de venta
-----------------------------------------------------------------------------------------------
"""
{
  'name': 'POS Extension for Bills', #Nombre del modulo
  'version': '1.0', #Versión en la que se encuentra el modulo
  'category': 'Sales/Point Of Sale', #Categoría de clasificación o dominio empresarial dentro de Odoo
  'author': 'Edgar Benedetto', #Autor del Moulo
  'depends': ['point_of_sale'], #Modulo que debe cargar antes del actual
  'data': [
      'views/category_total.xml'
  ], #Lista de rutas de archivos de datos que siempre deben instalarse o actualizarse con el módulo.
  'installable': True, #Modulo instalable
}
```

1. Archivo Descriptor del módulo **pos_extension** `__init__.py` .

   - Nombre del módulo: **pos_extension**
   - Nombre del archivo:

     `__init__.py`
   - Contenido del archivo:

**In [ ]:**

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------

    @author edgar.benedetto@unah.hn
    @date 02/07/2021
    @decription Archivo descriptor del módulo extensión de punto de venta
    @name_file __init__.py
    @version 1.0

-----------------------------------------------------------------------------------------------
    Archivo Descriptor del Módulo - Extensión del módulo de Punto de Venta
-----------------------------------------------------------------------------------------------
"""

# Importar los modelos creados
import . from models
import . from views
```

1. Dentro de la carpeta ***"models"*** crear el Archivo Descriptor del modelo correspondiente al módulo **pos_extension** `__init__.py` .

   - Nombre del módulo: **pos_extension**
   - Nombre del modelo: **category_total**
   - Nombre del archivo:

     `__init__.py`
   - Contenido del archivo:

**In [ ]:**

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------

    @author edgar.benedetto@unah.hn
    @date 02/07/2021
    @decription Archivo descriptor del modelo del módulo extensión de punto de venta
    @name_file __init__.py
    @version 1.0

-----------------------------------------------------------------------------------------------
    Archivo Descriptor del Modelo - Extensión del módulo de Ventas
-----------------------------------------------------------------------------------------------
"""
```

```
# Importar los modelos creados
import . from category_totals
```

1. Dentro de la carpeta ***"models"*** crear el Archivo del modelo **category_total** `category_total.py` .

   - Nombre del módulo: **pos_extension**
   - Nombre del modelo: **category_total**
   - Nombre del archivo:

     `category_total.py`
   - Contenido del archivo:

**In [ ]:**

```
# -*- coding: utf-8 -*-
"""
-----------------------------------------------------------------------------------------------------

    @author edgar.benedetto@unah.hn
    @date 02/07/2021
    @decription Modelo para agregar el subtotal de productos a partir de su categoria y obtener
      el total de todas las cantidades pedidas
    @name_file category_total.py
    @version 1.0

-----------------------------------------------------------------------------------------------------
    Modelo Total Categoria - Extensión del Punto en venta
-----------------------------------------------------------------------------------------------------
"""

class TotalByCategory(models.Model):
  """
  Obtener los subtotales de los productos contenidos en la factura del cliente a partir de
    su categoria para: MAQUILLAJE, REPELENTES, ALCOHOL, ESMALTES Y OTROS
  Obtener el total o la suma de los subtotales anteriores
  """
  _name = 'category_total'
  _inherit = 'pos.order'

  @api.depends('pos.order.line')
  def _compute_total_categories(self):
    """
    --> Obtener subtotales y almacenarlos por categoria dentro del objeto categorias
    --> Se recorren los elementos de la factura mientras se corrobora que el producto tenga alguna de las
        categorias solicitadas usando map y lambda
    --> El parametro trim = False permite que saltos de línea y espacios sean almacenados
      en el campo
    """
    _categories = {'MAQUILLAJE':0, 'REPELENTES':0, 'ALCOHOL':0, 'ESMALTES':0, 'OTROS':0}

    _total_categories_result = int(map(lambda _categories, _total_by_cats : map(_fill_categories(self, _c

    _subtotal_to_print = ("""
    Maquillaje: L {0}
    Repelentes: L {1}
    Alcohol: L {2}
    Esmaltes: L {3}
    Otros: L {4}
    """.format(float(_categories['MAQUILLAJE']), _categories['REPELENTES'], _categories['ALCOHOL'], _cate

    _total_to_print = ("""
    Total: L {}
    """.format(0))

    # Para que capture el campo
    _to_print = _subtotal_to_print + _total_to_print

    def _fill_categories(self, _categories, _total_by_cats):
      """
      Llenar el objeto categorias con los subtotales por categoria de los productos incluidos en la factu
      """
        for _category, _subtotal in _categories.items():
            _total_by_cats = self.product_id.price_unit * self.product_uom_qty
            if (self.product_id.display_name == _category):
              _categories[_category] +=  self.product_id.price_unit * self.product_uom_qty
```

```
            else:
              _categories['OTROS'] +=  self.product_id.price_unit * self.product_uom_qty
        return (_categories, _total_by_cats)

    # Campo computado o calculado a agregar en la vista que contiene todo lo solicitado por el cliente
    _total_result = fields.Char(string='Subtotal por Categorias', compute=_compute_total_categories, trim
```

1. **En el directorio *"views"* se debe crear el archivo :** `category_total.xml`

   - **Nombre del módulo: sales_extension**
   - **Nombre del modelo: category_total**

   - **Nombre del archivo:**

     `category_total.xml`
   - **Contenido del archivo:**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
    @author edgar.benedetto@unah.hn
    @date 02/07/2021
    @decription Vista para agregar los campos solicitados en la factura
       (subtotal categorias y total)
    @name_file category_total.xml
    @version 1.0

    ========================================================================
    Vista Factura - Extensión del Punto de Venta Factura de cliente
      -> Añadir los campos computados solicitados en la Factura
    ========================================================================
-->
<odoo>
  <data>
    <!-- Heredar la vista del producto -->
    <template name="report_invoice_document" ref="account.report_invoice_document"/>
      <table class="table table-sm" style="page-break-inside: avoid;">
        <tr class="border-black" style="">
          <td><strong>Subtotal y Total Por categoria</strong></td>
            <td class="text-right">
              <field name="_total_result"/>
              </field>
            </td>
          </td>
        </tr>
      </table>
    </template>
  </data>
</odoo>
```