

Escuela de Programación - Python B1

Proyecto Final: Sistema de comida rápida

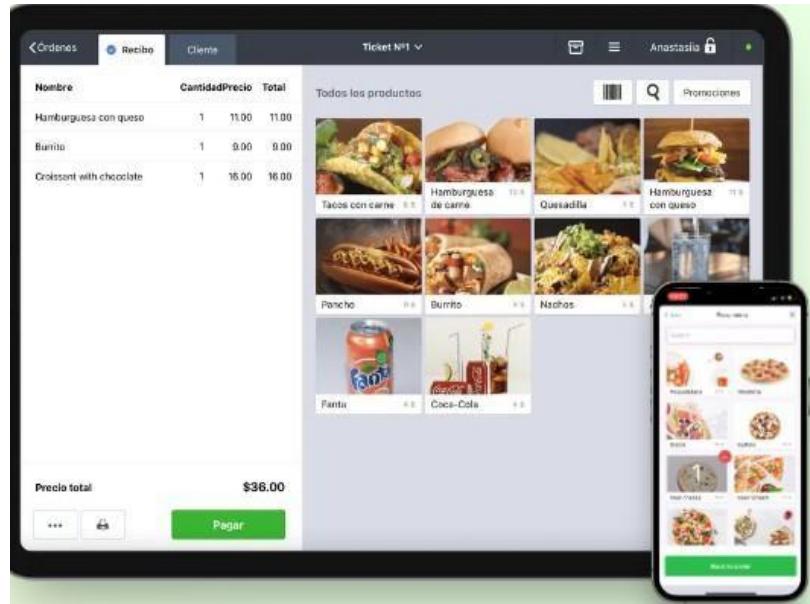
Introducción

Este documento describe el proyecto final de programación para el nivel B1 de la Escuela de Programación. El proyecto consiste en la implementación de un sistema de comida rápida. Para su desarrollo se ponen en práctica los conocimientos adquiridos durante el curso.

Criterios de evaluación

- Demostrar que se han asimilado correctamente los contenidos teóricos y que se aplican con precisión en una actividad de carácter práctico.
- Razonar todas las respuestas de forma analítica y sintética.
- Estructurar correctamente el código.
- Esquematizar las respuestas mediante el uso de estructuras de control condicionales y de flujos.
- Capacidad para convertir pseudocódigo en una implementación concreta.

Contexto



Fuente: Poster. Sistema Para Restaurantes de Comida Rápida, 2022

Según la Wikipedia: "El concepto de comida rápida es un estilo de alimentación donde el alimento se prepara y sirve para consumir rápidamente en establecimientos especializados".

En este ejercicio abordaremos la implementación de un sistema simple para procesar órdenes de un negocio de comida rápida. Para ello simularemos una base de datos utilizando varios archivos en formato CSV, donde cada uno tendrá la información de los cajeros, los clientes y los productos. El objetivo final es crear una aplicación que permita a un cajero preparar una orden para un cliente al ir agregando productos y calcular el monto total del pedido. Es importante notar que emplearemos varios paquetes o componentes que luego deberán integrarse entre sí.

Adicionalmente, los cajeros, clientes y productos deben ser representados como objetos, de esta manera se pondrá en práctica las técnicas del paradigma de la programación orientada a objetos.

 **Nota:** Este ejercicio tiene una serie de pistas para que puedas desarrollar y generar una solución con todo lo visto durante el desarrollo del curso.

Descripción de las actividades

Implementar un paquete llamado 'products' que tiene dos módulos: 'food_package.py' y 'product.py', con la siguiente estructura:

```
products/
    __init__.py
    food_package.py
    product.py
```

El módulo `food_package.py` contendrá una clase abstracta denominada 'FoodPackage' con dos funciones abstractas: 'def pack(self) -> str' y 'def material(self) -> str'. Esta clase nos permite crear un tipo específico de paquete o envoltura dependiendo del tipo de alimento a empacar, por ejemplo:

- Un vaso de soda puede ser empacado en un paquete tipo vaso y el material puede ser cartón.
- Una hamburguesa puede ser empacada en un paquete tipo envoltura de papel y el material puede ser aluminio.

En el mismo módulo se deberán incluir las implementaciones concretas para cada una de las siguientes clases 'Wrapping', 'Bottle', 'Glass' y 'Box', es decir, estas deben implementar los métodos anteriores y devolver un valor. Por ejemplo, la clase 'Wrapping' se puede definir como:

```
class Wrapping(FoodPackage):
    def pack(self):
        return "Food Wrap Paper"
    def material(self):
        return "Aluminium"
```

El módulo 'product.py' contendrá una clase abstracta denominada 'Product' con dos funciones abstractas: 'def type(self) -> str' y 'def foodPackage(self)-> FoodPackage'. Esta clase nos permita crear un producto específico y relacionarlo con su tipo de empaque por ejemplo:

- Un producto con código de barras G1, es una soda Sprite cuyo precio es de 5 euros, pertenece al tipo Soda y puede ser empacado en un paquete tipo vaso y el material puede ser cartón.
- Un producto con código de barras H1, es una hamburguesa Bacon cuyo precio es de 15 euros, pertenece al tipo Hamburger y puede ser empacado en un paquete tipo envoltura de papel y el material puede ser aluminio.

En el mismo módulo se deberán incluir las implementaciones concretas para cada una de las clases 'Hamburger', 'Soda', 'Drink' y 'HappyMeal', es decir, de forma parecida al módulo anterior, estas deben implementar los métodos anteriores y devolver un valor. Por ejemplo, la clase 'Hamburger', se puede definir como:

```
class Hamburger(Product):
```

```
def __init__(self, id:str, name:str, price:float):
    super().__init__(id, name, price)
def type(self) -> str:
    return "Hamburger"
def foodPackage(self) -> FoodPackage:
    return Wrapping()
```

Implementar un paquete llamado 'users' que tiene un módulo 'user.py', con la siguiente estructura:

```
users/
    __init__.py
    user.py
```

El módulo 'user.py' contendrá una clase abstracta denominada 'User' que tiene un constructor por defecto para los siguientes datos 'def __init__(self, dni:str, name:str, age:int)', con una función abstracta: 'def describe(self)'.

Luego en el mismo módulo se deberán incluir las implementaciones concretas para cada una de las clases 'Cashier' y 'Customer', es decir, estas deben implementar los métodos anteriores y devolver un valor. Adicionalmente, estas clases se diferencian por los parámetros que reciben sus constructores, por tanto, debemos hacer uso de herencia para inicializar el constructor de la clase padre y agregar características propias a cada clase.

Implementar un paquete llamado 'util' que tiene dos módulos, denominados 'file_manager.py' y 'converter.py', con la siguiente estructura:

```
util/
    __init__.py
    file_manager.py
    converter.py
```

El módulo 'file_manager.py' contendrá una clase 'CSVFileManager' la cual es una implementación libre y debe incluir las funciones:

- 1) La función 'def read(self)' lee un archivo en formato CSV y permite exportar su resultado como un Data Frame.
- 2) La función 'def write(self, DataFrame)' convierte un Data Frame en un archivo CSV. Esta es una función opcional, se deja al estudiante la implementación.

Los archivos en formato CSV se encuentran en la ruta "data/", a continuación, se describe el contenido de cada archivo:

- cashiers.csv: Información de los cajeros que harán uso del sistema.
- customers.csv: Información de los clientes que harán uso del sistema.
- drinks.csv: Información de los diferentes tipos de bebidas.
- sodas.csv: Información de los diferentes tipos de gaseosas.
- hamburgers.csv: Información de los diferentes tipos de hamburguesas.
- happyMeal.csv: Información de los diferentes tipos de happy meals.

El módulo 'converter.py' contendrá una clase denominada 'Converter' con una función abstracta para convertir las filas de un *Data Frame* en instancias de objetos. La función sería 'def convert(self, dataFrame, *args) -> list'. Adicionalmente esta clase debe incluir un método que permite imprimir la información de los objetos 'def print(self, list)'. En el mismo módulo se deberán incluir las implementaciones específicas que permitan leer los archivos en formato CSV y convertir sus filas en objetos de cada clase utilizando los paquetes product y users.

Implementar un paquete llamado 'orders' que tiene un módulo 'order.py', con la siguiente estructura:

```
orders/
    __init__.py
    order.py
```

El módulo 'order.py' contendrá una clase denominada 'Order' con un constructor 'def __init__(self, cashier:Cashier, customer:Customer):', el cual permite inicializar la clase con los datos del cajero, del cliente y la lista de productos vacía por defecto. Además, debe incluir tres funciones para agregar productos, calcular el total de la orden solicitada y mostrar la información de la orden que está siendo procesada. Las funciones son 'def add(self, product: Product)', ' def calculateTotal(self) -> float' y 'def show(self)', respectivamente.

Finalmente tendremos una clase principal que se llamará 'PrepareOrder' en la cual se deberá realizar una implementación que permita integrar los diferentes módulos empleados para leer los archivos en formato CSV y convertirlos en objetos. La implementación de esta clase es libre, es decir, no indicaremos las funciones que debe contener, pero la funcionalidad de la clase debe permitir crear una opción de menú que permita buscar los clientes, los cajeros y los productos para finalmente crear una orden.

Se sugiere utilizar los métodos de entrada de teclado para leer los datos del dni cajero, cliente e id de los productos.

A grandes rasgos, la aplicación seguiría los siguientes pasos:

- 1) Leer archivos en formato csv:
 - a. Leer cada archivo en formato csv: Utilizar una instancia de la clase 'CSVFileManager' y llamar al método 'read()'.
- 2) Convertir a listas de objetos:
 - a. Convertir cajeros: Utilizar una instancia la clase 'XxxConverter' y llamar al método 'convert()' e imprimir la lista llamando al método 'print()'.
 - b. Convertir clientes: Utilizar una instancia la clase 'XxxConverter' y llamar al método 'convert()' e imprimir la lista llamando al método 'print()'.
 - c. Convertir productos: Utilizar una instancia la clase 'XxxConverter' y llamar al método 'convert()' e imprimir la lista llamando al método 'print()'.
- 3) Preparar Orden:
 - a. Buscar cajero por dni: Función creada por el alumno y debe devolver una instancia de tipo cajero.

- b. Buscar cliente por dni. Función creada por el alumno y debe devolver una instancia de tipo cliente.
 - c. Inicializar Orden: Utilizar una instancia la clase 'Order', e inicializar con su constructor por defecto.
 - d. Mostrar productos a vender: Función creada por el alumno.
 - e. Escoger productos: Función creada por el alumno.
 - f. Agregar productos: Utilizar la instancia la clase 'Order', del paso c y llamar al método 'add()'.
- 4) Mostrar Orden: Utilizar la instancia la clase 'Order', del paso c y llamar al método 'show()'

Veamos un ejemplo de los pasos para dos productos:

- 1) Leer archivos en formato csv:
Ejemplo código:
`df_cashiers = CSVFileManager("ruta/cashier.csv").read()`
- 2) Convertir a listas de objetos
 - a. Convertir cajeros:
Ejemplo código:
`converter = XxxConverter()
cashier_list = converter.convert(df_cashiers)
converter.print(cashier_list)`
Ejemplo consola:
`Cashier - Name: Mia, DNI: 5001 , Timetable: 09:00-15:00, Salary: 800.0.
Cashier - Name: Sinthy, DNI: 5002 , Timetable: 09:00-14:00, Salary: 750.6.
Cashier - Name: Carles, DNI: 5003 , Timetable: 11:00-17:00, Salary: 655.3.`
- 3) Preparar Orden
 - a. Buscar cajero por dni: Función creada por el alumno.
Ejemplo consola:
`Introduce DNI cashier: 5001`
`Cashier - Name: Mia, DNI: 5001 , Timetable: 09:00-15:00, Salary: 800.0.`
 - b. Buscar cliente por dni. Función creada por el alumno.
Ejemplo consola:
`Introduce customer DNI: 1001`
`Customer - Name: Mike, DNI: 1001 , Age: 37, Email: mike@uoc.edu, Postal
Code: 28009`
 - c. Inicializar Orden: Utilizar una instancia la clase 'Order', e inicializar con su constructor por defecto.
Ejemplo código:
`order = Order(cashier, customer)`
 - d. Mostrar productos a vender: Función creada por el alumno.
Ejemplo consola:
`Product list:`
`Product - Type: Hamburger, Name: Mc Bacon, Id: H1 , Price: 10.8 , Wrapping:
Food Warp Paper , Material: Aluminium.`
`Product - Type: Hamburger, Name: Super Delux, Id: H2 , Price: 15.6 , Wrapping:
Food Warp Paper , Material: Aluminium.`

- Product - Type: Soda, Name: Coca Cola, Id: G1 , Price: 1.5 , Wrapping: Bottle, Material: Plastic.
- Product - Type: Soda, Name: Seven Up, Id: G3 , Price: 1.1 , Wrapping: Bottle, Material: Plastic.
- e. Escoger productos: Función creada por el alumno.
Ejemplo consola:
Introduce product id: H1
Product - Type: Hamburger, Name: Mc Bacon, Id: H1, Price: 10.8 , Wrapping: Food Warp Paper, Material: Aluminium.
Do you want to add another product?: Yes
Introduce product id: G1
Product - Type: Soda, Name: Coca Cola, Id: G1 , Price: 1.5 , Wrapping: Bottle, Material: Plastic.
Do you want to add another product?: No
- f. Agregar productos: Utilizar la instancia la clase 'Order', del paso c y llamar al método 'add()' .
Ejemplo código:

```
for product in selected_product_list:  
    order.add(product)
```
- 4) Mostrar Orden: Utilizar la instancia la clase 'Order', del paso c y llamar al método 'show()'
Ejemplo código:

```
order.show()
```


Ejemplo consola:
Hello: Customer - Name: Mike, DNI: 1001 , Age: 37, Email: mike@uoc.edu, Postal Code: 28009
Was attended by: Cashier - Name: Mia, DNI: 5001 , Timetable: 09:00-15:00, Salary: 800.0.
Product 1: Product - Type: Hamburger, Name: Mc Bacon, Id: H1 , Price: 10.8 , Wrapping: Food Warp Paper, Material: Aluminium.
Product 2: Product - Type: Soda, Name: Coca Cola, Id: G1 , Price: 1.5 , Wrapping: Bottle, Material: Plastic.
Total price: 12.3

Nota: Opcionalmente se puede implementar la función 'write' de la clase 'CSVFileManager' y exportar las órdenes hacia un archivo nuevo con formato csv, el cual debe contener los siguientes campos de cada orden:

- 1) DNI cajero
- 2) DNI comprador
- 3) Fecha y hora de la venta
- 4) Total

Evaluación

La puntuación de cada pregunta es la siguiente:

- Pregunta 1: 100%

Se valorará la validez de la solución y la claridad de la argumentación.

El ejercicio tiene indicado en el enunciado su peso en la valoración final. Los criterios de evaluación para evaluar este ejercicio son los siguientes:

Pregunta	No logrado (C-)	Mínimamente logrado (C+)	Logrado (B)	Logrado de forma excelente (A)
Pregunta 1	La respuesta es incorrecta o no fue desarrollada	La respuesta es parcialmente correcta y está mínimamente justificada	La respuesta es correcta y describe ciertos pasos para llegar a la solución, y está justificada	La respuesta es correcta e indica todos los pasos para llegar a la solución. Se referencia correctamente el artículo y apuntes para justificar la respuesta

Formato

Se sugiere, con el objetivo de estandarizar el formato, la actividad siga las siguientes restricciones:

- Se debe entregar el código desarrollado a través del aula Canvas en formato .zip.
- Se pueden realizar varias entregas en la plataforma.
- El código desarrollado debe incluir comentarios en el código.
- El estudiante debe notificar al Docente Colaborador un mensaje una vez que considere que el ejercicio esté listo para la calificación.
- De manera opcional, se puede usar un repositorio de GitHub con el correo de la UOC para subir la solución. En ese caso, se debe:
 - Poner el repositorio en público para que el Docente Colaborador pueda acceder a él.
 - Incluir la URL del repositorio en el mensaje.
 - Usar el repositorio de GitHub no excluye la obligación de entregar el código a través del aula Canvas.

Entrega

La entrega de la Actividad deberá realizarse en la sección: **Actividad Proyecto Final del aula Canvas**.