-------------------------------------------------------

CREER UNE CLASSE PIZZAS :

-------------------------------------------------------  
# Nom, prix ,ingrédients, végétarienne (ou non)  
  
class Pizza: # Le nom d'une classe s'écrit avec une majuscule  
 # Chaque pizza aura un nom et un prix différents -> variable d'instance à mettre dans le constructeur.  
 def \_\_init\_\_(self, nom, prix, ingredients, vegetarienne=False): # False par défaut car True est indiqué si elle l'est  
 # On définit le constructeur et ajoute des paramètres nom, prix, ingredients et végétarienne (ou non)  
 self.nom = nom # Variable d'instance  
 self.prix = prix  
 self.ingredients = ingredients  
 self.vegetarienne = vegetarienne

# AFFICHER LE NOM ET LE PRIX DE LA PIZZA :  
 def Afficher(self):  
 veg\_str = "" # Rien par défaut  
 if self.vegetarienne:  
 veg\_str = " - VEGETARIENNE" # Si la pizza est végétarienne, on ajoute l'info ci-dessous  
 print(f"PIZZA {self.nom} : {self.prix} €" + veg\_str + ".")  
 print(", ".join(self.ingredients))  
 print()  
  
----------------------------------------------------------  
CREER UNE PIZZA PERSONNALISEE :

----------------------------------------------------------  
class PizzaPersonnalisee(Pizza): # C'est une pizza enfant de Pizza  
 PRIX\_DE\_BASE = 7  
 PRIX\_PAR\_INGREDIENT = 1.2  
 dernier\_numero = 0 # Ajouter automatiquement plusieurs pizzas perso  
  
 def \_\_init\_\_ (self): # Constructeur  
 PizzaPersonnalisee.dernier\_numero += 1 # Incrémenter pour toutes les classes  
 self.numero = PizzaPersonnalisee.dernier\_numero # Numéro ici car on en aura besoin + bas  
 super().\_\_init\_\_("Personnalisée " + str(self.numero), 0, []) # Titre  
 # Appel du constructeur du parent  
 self.demander\_ingredients\_utilisateur()  
 self.calculer\_le\_prix()  
  
 def demander\_ingredients\_utilisateur(self):  
 print()  
 print(f"Ingrédients pour la pizza personnalisée {self.numero}.")  
 while True: # Toujours boucler  
 ingredient = input("Ajoutez un ingrédient (ou ENTER pour terminer) : ")  
 if ingredient == "":  
 return  
 self.ingredients.append(ingredient)  
 print(f"Vous avez {len(self.ingredients)} ingrédient(s) : {', '.join(self.ingredients)}.")  
 # Afficher le nbre d'ingredients.  
  
 CALCUL DU PRIX :  
 def calculer\_le\_prix(self):  
 self.prix = self.PRIX\_DE\_BASE + len(self.ingredients)\*self.PRIX\_PAR\_INGREDIENT

--------------------------------------------------------  
CREATION DE LA PREMIERE PIZZA :

--------------------------------------------------------  
pizza1 = Pizza("4 fromages", 8.5, ("brie", "emmental", "conté", "parmesan"))  
# Création d'un objet Pizza, stockée dans la variable pizza1. Possède ses caractéristiques nom/prix en paramètres  
# Possède une collection (tuple) en 3eme paramètre.  
  
pizza1.Afficher()  
  
  
FABRIQUER UNE LISTE DE PIZZAS :  
pizzas = [  
 Pizza("4 Fromages", 8.5, ("Brie", "Emmental", "Conté", "Parmesan"), True),  
 Pizza("Hawai", 8.5, ("Tomate", "Ananas", "Oignons")),  
 Pizza("4 Saisons", 11, ("Oeufs", "Emmental", "Tomates", "Jambon")),  
 Pizza("Végétarienne", 7.8, ("Champignons", "Tomates", "Oignons", "Poivrons"), True),  
 # True -> paramètre optionnel pour indiquer que cette pizza est végétarienne.  
 PizzaPersonnalisee(),  
 PizzaPersonnalisee() # Plusieurs pizzas personnalisées  
]  
  
TRIER LES PIZZAS :  
# Permet de donner un critère de comparaison  
# car Python ne sait pas comment trier une liste de pizzas qui comprend ici des chaines et des nombres  
def tri(e): # Cette définition prend en paramètre un élément e  
 return e.nom # ici, e = pizza  
 # On peut remplacer nom par prix par exemple.  
 # return len(e.ingredients)  
 *"""*

*Attention, on ne peut pas appeler la fonction sort sur un tuple car sort modifie l'ordre des données.  
# Il faut donc faire une liste -> []  
# pizzas.sort(key=tri, reverse=True) # key=tri -> critère de comparaison*

*"""*  
  
# AFFICHER CETTE LISTE DE PIZZAS EN FONCTION DE CONDITIONS :  
for i in pizzas: # Boucle for pour afficher toutes les pizzas  
 i.Afficher()  
 # if i.vegetarienne: # Afficher uniquement les pizzas végétariennes  
 # i.Afficher() # i dans la collection pizzas va récupérer les différents éléments de cette collection  
 # if not i.vegetarienne: # Afficher uniquement les pizzas non végétariennes  
 # i.Afficher()  
 # if "Tomates" in i.ingredients: # Afficher uniquement les pizzas qui ont de la tomate  
 # i.Afficher()  
 # if i.prix < 10: # Afficher uniquement les pizzas à moins de 10 euros  
 # i.Afficher()  
  
  
# AMELIORER L'AFFICHAGE DES INGREDIENTS :  
ingredients = ("brie", "emmental", "conté", "parmesan")  
print(", ".join(ingredients))  
# Ici, la programme va prendre chaque valeur et les séparer par un délimiteur : la ", ".

*"""  
CONCLUSION :  
Démarrage de la POO.  
1) Création de la class pizza avec # propriétés comme le nom, prix, etc --> variables d'instance,  
et on a passé les différentes valeurs dans le constructeur.  
  
2) Création de la fonction Afficher, qui est une méthode de la class pizza.  
A l'intérieur, on s'est basé sur les variables d'instance, en utilisant le mot-clé self.  
  
3) Création de la pizza personnalisée grâce à l'héritage. Elle a hérité de ses propriétés, et de la fonction Afficher.  
On a ajouté des propriétés supplémentaires comme PRIX\_DE\_BASE   
des fonctionnalités supplémentaires (demander\_ingredients\_utilisateur) + méthodes (calculer\_le\_prix).  
"""*