Estructuras de datos _

Listas

Contenedores que permiten almacenar un conjunto de datos en una misma variable. Pueden ser datos de el mismo o de distinto tipo.

Funciones de listas (doc)

- list.**append**(*x*): Agrega un elemento al final de la lista.
- list.**insert**(*i*, *x*): Inserta un elemento en la lista en la posición indicada.
- list.**remove**(*x*): Elimina de la lista la primera ocurrencia del elemento indicado.
- list.**pop**([*i*]): Saca el último elemento de la lista, o el del índice indicado, y lo retorna.
- list.**sort**(*key=None*, *reverse=False*): Ordena los elementos de la lista ascendentemente.
- list.**reverse**(): Revierte el orden de los elementos de la lista.

Desafío - Funciones de listas

pizza.py

Paso 1: Definir ingredientes base

```
1  # Ingredientes "base"
2  ingredientes = ["masa tradicional", "salsa de tomate", "queso"]
3  opcion = 1
4
```

Paso 2: Definir ciclo while y opciones

```
2  ingredientes = ["masa tradicional", "salsa de tomate", "queso"]
3  opcion = 1
4
5  while opcion < 6:
6      print("¡Gracias por ordenar con nosotros!, ¿Qué desea realizar?")
7      print("1. Consultar ingredientes de la pizza")
8      print("2. Cambiar tipo de masa")
9      print("3. Cambiar tipo de salsa")
10     print("4. Agregar ingredientes")
11     print("5. Eliminar ingredientes")
12     print("6. Ordenar")
13
14     # Guardar opción como int
15     opcion = int(input())
16
17     if opcion == 1:
18
19     elif opcion == 2:
20
21     elif opcion == 3:
22
23     elif opcion == 4:
24
25     elif opcion == 5:
26
```

Paso 3: Opción 1 - Mostrar ingredientes

```
17      if opcion == 1:
18          # Mostrar la lista de ingredientes
19          print("Ingredientes seleccionados:", ingredientes)
20          print()
21
```

Paso 4: Opción 2 - Definir flujo de acción

```
19    elif opcion == 2:
20
21        # 1. Consultar si entre los ingredientes está la masa tradicional
22            # 1.1 Si está, consultar si la desea cambiar por masa delgada
23                # 1.1.1 Si la quiere cambiar, eliminar de la lista la masa tradicional, y agregar masa delgada
24                # 1.1.2 Si no la quiere cambiar, volver a mostrar el menú
25
26        #1.2 Si no está, consultar si tiene en la lista la masa delgada
27            # 1.2.1 Si está, consultar si la desea cambiar por masa tradicional
28                # 1.2.1.1 Si dice que sí, eliminar de la lista la masa delgada, y agregar masa tradicional
29                # 1.2.1.2 Si dice que no, volver al menú
30
31        # 1.3 No tiene ningún tipo de masa (else)
32            # 1.3.1 Consultar con un nuevo submenu qué masa desea: 1. Tradicional, 2. Delgada
33
```

Paso 5: Opción 2 - Caso 1: Flujo

```
# 1. Consultar si entre los ingredientes está la masa tradicional
if "masa tradicional" in ingredientes:

    # 1.1 Si está, consultar si la desea cambiar por masa delgada
    print("¿Cambiar masa tradicional por masa delgada?")
    print("Escriba 'S' para cambiar, o 'N' para conservar la masa tradicional")
    print()
    cambiar = input().upper()

    if cambiar == 'S':
        # 1.1.1 Si la quiere cambiar, eliminar de la lista la masa tradicional, y agregar masa delgada

        # Eliminar masa tradicional con "remove"
        ingredientes.remove("masa tradicional")

        # Agregar masa delgada. Solo por temas de orden,
        # utilizaremos 'insert' para agregarla en la posición 0
        ingredientes.insert(0, "masa delgada")


    # 1.1.2 Si no la quiere cambiar, volver a mostrar el menú
```

Paso 6: Opción 2 - Caso 2: Flujo

```
#1.2 Si no está, consultar si tiene en la lista la masa delgada
elif "masa delgada" in ingredientes:

    # 1.2.1 Si está, consultar si la desea cambiar por masa tradicional
    print("¿Cambiar masa delgada por tradicional?")
    print("Escriba 'S' para cambiar, o 'N' para conservar la masa delgada")
    print()
    cambiar = input().upper()

    if cambiar == "S":
        # 1.2.1.1 Si dice que sí, eliminar de la lista la masa delgada, y agregar masa tradicional

        # Eliminar masa delgada con "remove"
        ingredientes.remove("masa delgada")

        # Agregar masa tradicional. Solo por temas de orden,
        # Utilizaremos "insert" para agregarla en la posición 0
        ingredientes.insert(0, "masa tradicional")

    # 1.2.1.2 Si dice que no, volver al menú
```

Paso 7: Opción 2 - Caso 3: Flujo

```
# 1.3 No tiene ningún tipo de masa (else)
else:

    # 1.3.1 Consultar con un nuevo submenu qué masa desea: 1. Tradicional, 2. Delgada
    print("¡Su pizza no tiene masa!")
    print("Escriba '1' si desea masa tradicional, o '2' si desea masa delgada")
    print()

    masa = input()

    # Se agrega opción escogida con insert
    if masa == "1":
        ingredientes.insert(0, "masa tradicional")
    elif masa == "2":
        ingredientes.insert(0, "masa delgada")
```

Paso 8: Opción 2 - Caso 4: Flujo

```
85      elif opcion == 4:
86          #Lista de ingredientes
87          print("¿Qué ingrediente desea agregar?")
88          print("Tomate")
89          print("Aceitunas")
90          print("Queso")
91          print("Peperoni")
92          print("Pollo")
93          print()
94
95          nuevo = input().lower()
96
97          # Validar que sea un ingrediente permitido
98          while nuevo != "tomate" and nuevo != "aceitunas" and nuevo != "queso" and nuevo != "peperoni" and nuevo != "pollo":
99              print("Ingrese un ingrediente válido")
100             print("Tomate")
101             print("Aceitunas")
102             print("Queso")
103             print("Peperoni")
104             print("Pollo")
105             print()
106
107             nuevo = input().lower()
108
109         # Agregar ingrediente al final de la lista
110         ingredientes.append(nuevo)
```

Paso 9: Opción 2 - Caso 5: Flujo

```
112     elif opcion == 5:
113         # Consultamos ingrediente para eliminar
114         print("Ingredientes actuales:", ingredientes)
115         print("Escriba el que desea eliminar")
116         print()
117         eliminar = input().lower()
118 
119         # Validación
120         while eliminar not in ingredientes:
121             print("Ingredientes actuales:", ingredientes)
122             print("Escriba el que desea eliminar")
123             print()
124             eliminar = input().lower()
125 
126         # Eliminar con remove
127         ingredientes.remove(eliminar)
```

Sugerencias de mejoras

1. Validaciones para que dentro de las opciones principales solo se escojan de la 1 a la 6.

2. Validaciones para que dentro de las opciones que lo requieran, solo se ingresen los campos permitidos.

3. Agregar más mensajes de print para ayudar en el flujo al usuario.

4. Replicar lógica de las masas con la salsa (salsa de tomate o salsa BBQ).

5. Crear otra lista con los “ingredientes base”, y utilizar esta lista para mostrar los ingredientes disponibles, y validar la entrada del usuario.

6. Consultar por agregar o eliminar ingredientes continuamente (sin tener que volver al menú principal) hasta que el usuario indique que ya no desea hacer modificaciones.

7. Contar la frecuencia de cada ingrediente.

8. Ordenar los ingredientes alfabéticamente, omitiendo la masa y la salsa que deben salir al comienzo.

9. Omitir masa y salsa en la lista de ingredientes para eliminar.

10. Asignar un precio a cada ingrediente y calcular el valor final de la pizza, y mostrarlo al escoger la opción “ordenar”.

Desafío - Transformaciones y filtros

tienda.py

Lista de artículos

artículos = [“celular”, “LG K10”, “90000”, “tablet, “Galaxy TAB”, “80000”, “smart tv”, “LED 43 Samsung”, “485000”, “celular”, “Galaxy J7”, “120000”, “celular”, “Huawei Y5”, “59900”, “notebook”, “Lenovo ideapad”, “250000”, “tablet”, “Huawei media”, “139000”, “notebook”, “Acer”, “145000”]

Objetivos

- Separar los precios en listas diferentes según categorías.
- Aplicar el descuento de 10% a los artículos correspondientes (que no sean notebooks, y que su precio sea mayor a $80.000)

Paso 1: Definir variables de inicio

```
articulos = ["celular", "LG K10", "90000",
             "tablet",  "Galaxy TAB", "80000",
             "smart tv", "LED 43 Samsung", "485000",
             "celular", "Galaxy J7", "120000",
             "celular", "Huawei Y5", "59900",
             "notebook", "Lenovo ideapad", "250000",
             "tablet", "Huawei media", "139000",
             "notebook", "Acer", "145000"]

celulares = []
notebooks = []
smart_tv = []
tablets = []
aux = ""

```

Paso 2: Definir loop

```
16  # Se iterará sobre los índices y los elementos, por ello se usa enumerate
17  for index, elemento in enumerate(articulos):
18
```

Paso 3: Definir categoría de artículo y precios

```
16  # Se iterará sobre los índices y los elementos, por ello se usa enumerate
17  for index, elemento in enumerate(articulos):
18
19      if index % 3 == 0:
20          # Será categoría
21      elif (index - 2) % 3 == 0:
22          # Será precio
23
```

Paso 4: Almacenar categoría en variable auxiliar

```
16  # Se iterará sobre los índices y los elementos, por ello se usa enumerate
17  for index, elemento in enumerate(articulos):
18
19      if index % 3 == 0:
20          aux = elemento
21      elif (index - 2) % 3 == 0:
```

Paso 5: Almacenar precio final según requerimiento

```
16  # Se iterará sobre los índices y los elementos, por ello se usa enumerate
17  for index, elemento in enumerate(articulos):
18
19      if index % 3 == 0:
20          aux = elemento
21      elif (index - 2) % 3 == 0:
22          int_precio = int(elemento)
23          precio_final = int_precio
24
25          if precio_final > 80000 and aux is not "notebook":
26              precio_final = int(precio_final * 0.9)
```

Paso 6: Almacenar precio en lista correspondiente

```
16  # Se iterará sobre los índices y los elementos, por ello se usa enumerate
17  for index, elemento in enumerate(articulos):
18
19      if index % 3 == 0:
20          aux = elemento
21      elif (index - 2) % 3 == 0:
22          int_precio = int(elemento)
23          precio_final = int_precio
24
25          if precio_final > 80000 and aux is not "notebook":
26              precio_final = int(precio_final * 0.9)
27
28          if aux == "celular":
29              celulares.append(precio_final)
30          elif aux == "notebook":
31              notebooks.append(precio_final)
32          elif aux == "smart tv":
33              smart_tv.append(precio_final)
34          elif aux == "tablet":
35              tablets.append(precio_final)
36
```

Paso 7: Ordenar precios de mayor a menor

```
37  celulares.sort()
38  celulares.reverse()
39
40  notebooks.sort()
41  notebooks.reverse()
42
43  smart_tv.sort()
44  smart_tv.reverse()
45
46  tablets.sort()
47  tablets.reverse()
48
49  print("Celulares:", celulares)
50  print("Notebooks:", notebooks)
51  print("Smart TV:", smart_tv)
52  print("Tablets:", tablets)
```

Código completo

```python
articulos = ["celular", "LG K10", "90000",
             "tablet",  "Galaxy TAB", "80000",
             "smart tv", "LED 43 Samsung", "485000",
             "celular", "Galaxy J7", "120000",
             "celular", "Huawei Y5", "59900",
             "notebook", "Lenovo ideapad", "250000",
             "tablet", "Huawei media", "139000",
             "notebook", "Acer", "145000"]

celulares = []
notebooks = []
smart_tv = []
tablets = []
aux = ""

# Se iterará sobre los índices y los elementos, por ello se usa enumerate
for index, elemento in enumerate(articulos):

    if index % 3 == 0:
        aux = elemento
    elif (index - 2) % 3 == 0:
        int_precio = int(elemento)
        precio_final = int_precio

        if precio_final > 80000 and aux is not "notebook":
            precio_final = int(precio_final * 0.9)

        if aux == "celular":
            celulares.append(precio_final)
        elif aux == "notebook":
            notebooks.append(precio_final)
        elif aux == "smart tv":
            smart_tv.append(precio_final)
        elif aux == "tablet":
            tablets.append(precio_final)
celulares.sort()
celulares.reverse()

notebooks.sort()
notebooks.reverse()

smart_tv.sort()
smart_tv.reverse()

tablets.sort()
tablets.reverse()

print("Celulares:", celulares)
print("Notebooks:", notebooks)
print("Smart TV:", smart_tv)
print("Tablets:", tablets)
```

Operaciones funcionales en listas

- Son otra forma para filtrar, transformar y reducir datos
- Su sintaxis es más simple y resumida que la de un ciclo for
- Las más utilizadas son map, filter y reduce

Pandas

Pandas

- Librería orientada a la manipulación y limpieza de estructuras de datos, que nos permite trabajar fácilmente con tablas de datos.
- Combina el alto desempeño de operaciones vectorizadas de numpy con las manipulaciones flexibles de hojas de cálculo y bases de datos relacionales.
- Muy utilizada en la comunidad.
- Se añade como “import pandas as pd”.

DataFrame a partir de csv

- Se utiliza función “read_csv”.
- “read_csv” retorna una estructura de datos “DataFrame”, que corresponde a una matriz con filas y columnas.
- “read_csv” recibe como parámetro un string con la ruta del archivo.

Serie

- Fila o columna de datos del DataFrame.
- Conjunto de asociaciones llave:valor.
- En el caso de una Serie como columna del dataframe, la llave de cada elemento será el índice de la fila.
- Se accede a una serie como columna del DataFrame escribiendo el nombre de la columna entre comillas dentro de paréntesis de corchetes.

Numpy

¿Qué nos permite?

- Crear listas o arreglos n-dimensionales, que facilitarán el trabajo aritmético.
- Utilizar funciones matemáticas de rápida velocidad, sin la necesidad de usar loops.
- Trabajar con álgebra lineal.
- Generar números pseudoaleatorios.
- Utilizar funciones distributivas.

Funciones que generan ndarrays

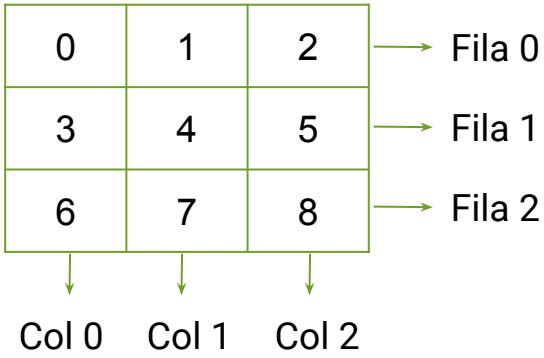
- **np.asarray**: Convierte inputs sólo si éstos no son ndarray.
- **np.arange**: Genera un rango de manera similar a la implementación nativa de range(), devolviendo una lista en formato ndarray.
- **np.ones, np.ones_like**: Generan arrays poblados de 1 con las dimensiones especificadas. np.ones_like toma como referencia las dimensiones y tipo de dato de otro objeto.
- **np.zeros, np.zeros_like**: Generan arrays poblados de 0 con las dimensiones especificadas. np.zeros_like toma como referencia las dimensiones y tipo de dato de otro objeto.
- **np.empty, np.empty_like**: Generan arrais poblados mediante la asignación de memoria, pero no completa los arrays con elementos.
- **np.full, np.full_like**: Produce un array con determinadas dimensiones y tipo de dato, rellenadas con un valor determinado.

Array n-dimensional

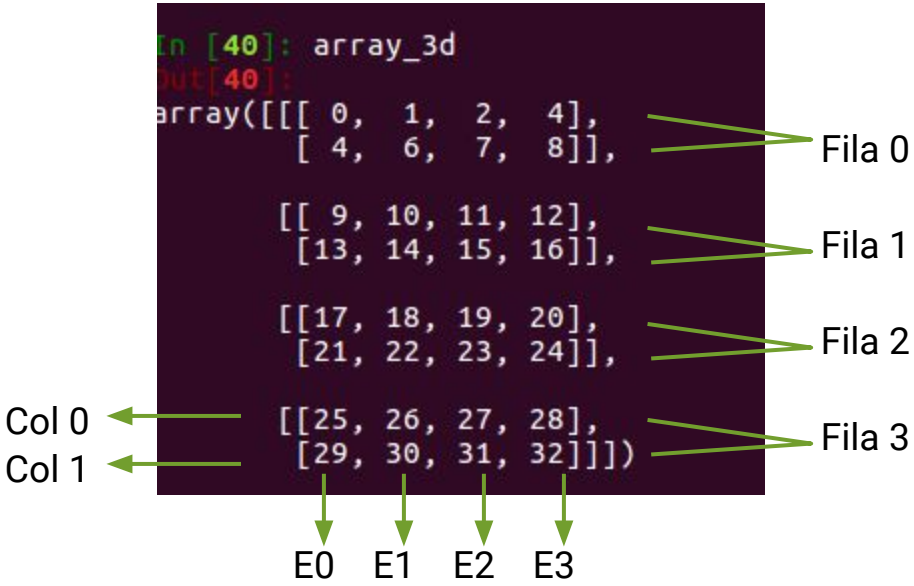
```
matriz = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
```

```
In [28]: matriz = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])

In [29]: matriz
Out[29]:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```



```
array_3d = np.array([ [ [0, 1, 2, 4], [4, 6, 7, 8] ], [ [9, 10, 11, 12],
[13, 14, 15, 16] ], [ [17, 18, 19, 20], [21, 22, 23, 24] ], [ [25, 26,
27, 28], [29, 30, 31, 32] ] ])
```



[ ]: 4 Filas, primera dimensión
[ ]: 2 Columnas por fila, segunda dimensión
26: 4 elementos dentro de cada columna dentro de cada fila, tercera dimensión