Ejercicios Propuestos - Control de flujo y ciclos

Lectura - Control de Flujo

Ejercicio Propuesto 1: Clasificar Password

Crear un programa donde el usuario debe ingresar un password en la plataforma. Si el password tiene menos de 6 letras, se debe mostrar el aviso "El password es demasiado corto".

Ejercicio Propuesto 2: Password Incorrecto

Crear un programa donde el usuario debe ingresar un password. Si el password es 12345, entonces se debe informar que el password es incorrecto.

Ejercicio Propuesto 3: Seguridad

La seguridad en las contraseñas es un tema importante, ya que dependiendo de la complejidad que tiene dicha contraseña pueden depender datos de extrema sensibilidad.

Una persona le dice que ha generado dos códigos para clasificar las contraseñas, pero no logra entender cuál es la diferencia entre cada uno y cuál es el que él necesita. Por ello, el objetivo final de la aplicación es clasificar la contraseña según los siguientes criterios:

- 4 letras o menos será corta.
- 5 a 10 letras será mediana.
- Más de 10 letras será larga.

NOTA: La librería `getpass` introduce el método `getpass.getpass()`. Este método funciona exactamente igual a `input()`. La diferencia es que puede ser más apropiado para el ingreso de claves ya que aparece oculta.

a)

```
import getpass
password = getpass.getpass("Ingrese una Contraseña")
largo = len(password)

if largo <= 4:
    print("Pequeña")
elif largo < 10:
    print("Mediana")
```

```
else:
    print("Larga")
```

b)

```
import getpass
password = getpass.getpass("Ingrese una Contraseña")
largo = len(password)

if largo <= 4:
    print("Pequeña")

if largo < 10:
    print("Mediana")
else:
    print("Larga")
```

- Determinar qué versión del código es la que entrega los resultados esperados.
- Modificar el código escogido para poder distinguir palabras muy largas, cuando éstas tengan 15 o más caracteres.

Lectura - El Ciclo While

Ejercicio Propuesto 1: Sumando de 1 a 100

$$1 + 2 + 3 + \ldots + 100 = ¿?$$

Resolver esto es muy similar a contar cien veces, pero además de contar, se debe ir guardando la suma acumulada en cada iteración.

Tips:

- Itera 100 veces.
- En cada iteración, utiliza un contador para almacenar la iteración.
- En cada iteración, asegúrate de utilizar un acumulador para acumular tus contadores.
- El acumulador luego de terminado el ciclo, corresponderá al resultado de la suma.
- Asegúrate de iniciar el contador y el acumulador antes de entrar al ciclo.

Ejercicio Propuesto 2: Generar una lista en HTML

Se pide crear un programa donde el usuario ingrese un número como argumento y se genere una lista de HTML con esa cantidad de ítems.

```
python lista_html.py 5
```

```
<ul>
    <li> 1 </li>
    <li> 2 </li>
    <li> 3 </li>
    <li> 4 </li>
    <li> 5 </li>
</ul>
```

Tips:

- Puedes tabular con `"\t"`.
- Puedes hacer un salto de línea con `"\n"`.
- También es posible evitar el uso de caracteres especiales utilizando strings multilínea (`"""`).

Lectura - Python Comprehensions

Ejercicio propuesto 1: Transformar ciclos en List Comprehensions

A continuación, te presentamos dos ejercicios a partir de los cuales deberás transformar ciclos en List Comprehensions siguiendo los pasos anteriormente explicados:

Transformar a Mayúsculas

```
mascotas = ['gato',
 'perro',
 'tortuga',
 'gato',
 'perro',
 'tortuga',
 'gato',
 'perro',
 'tortuga',
 'gato',
 'perro',
 'tortuga',
 'hurón',
 'hamster',
 'erizo de tierra']
```

```
mascotas_mayusculas = []

for mascota in mascotas:
    mascotas_mayusculas.append(mascota.upper())

print(mascotas_mayusculas)
```

Elevar al cuadrado

```
valores = [0,4,5,6,7,8,9]

lista_square = []
for valor in valores:
      lista_square.append(valor**2)

print(lista_square)
```

Ejercicio propuesto 2: Mayúscula solo Gatos

¿Qué pasa si para la lista de mascotas definida anteriormente se quiere transformar en mayúscula sólo a los gatos? La solución con `for` sería la siguiente:

```
mascotas_mayusculas = []

for i in mascotas:
    if i == 'gato':
        mascotas_mayusculas.append(i.upper())
    else:
        mascotas_mayusculas.append(i.lower())

print(mascotas_mayusculas)
```

1. ¿Cómo se escribiría este código como un List Comprehension?

Ejercicio propuesto 3: Filtrado

A continuación, se muestra cómo crear un programa que filtre todos los números de una lista que sean menores a 1000. Esto es lo mismo que decir "**seleccionar todos los elementos mayor o iguales a mil**".

1. Transformarlo en un List Comprehension

```
a = [100, 200, 1000, 5000, 10000, 10, 5000]
n = len(a)
filtered_array = []

for i in range(n):
    if a[i] >= 1000:
        filtered_array.append(a[i])

print(filtered_array)
```

```
[1000, 5000, 10000, 5000]
```

Ejercicio propuesto 4: Adictos a redes

Se tiene una lista con la cantidad de minutos usados en redes sociales de distintos usuarios.

```
[120, 50, 600, 30, 90, 10, 200, 0, 500]
```

1. Se debe retornar una lista que clasifique todos los tiempos menores a 90 minutos como 'bien' y todos los tiempos mayores o iguales a 90 como 'mal'.

El output debería ser algo similar a lo siguiente:

```
['mal', 'bien', 'mal', 'bien', 'bien', 'bien', 'mal', 'bien', 'mal']
```

Ejercicio propuesto 5: Adictos v2

Se pide utilizar la misma lista de minutos del ejercicio anterior.

1. Se debe retornar una nueva lista cambiando todas las medidas inferiores a 90 minutos como 'bien', entre 90 y 180 como 'mejorable', y todas las mayores o iguales a 180 como 'mal'.

```
['mejorable', 'bien', 'mal', 'bien', 'mejorable', 'bien', 'mal', 'bien',
'mal']
```

Ejercicio propuesto 6: Transformando segundos a minutos

Se tiene una lista con la cantidad de segundos que demoraron algunos procesos.

1. Se necesita una función para transformar todos los datos a minutos, (se requiere sólo la parte entera, la parte decimal se puede ignorar).

```
seconds = [100, 50, 1000, 5000, 1000, 500]
```

Ejercicio propuesto 7: Países

Tenemos un listado de países y la cantidad de usuarios por cada país en la siguiente tabla:

País	Cantidad de usuarios
México	70
Chile	50
Argentina	55

Tabla 1. Listado de países.
Fuente:

1. Convertir la tabla mostrada en un diccionario.
2. Filtrar los países con menos de 60 usuarios.

Ejercicio propuesto 8: Ventas

Dada la información de ventas de 3 meses:

Mes	Ventas
Octubre	65000
Noviembre	68000
Diciembre	72000

Tabla 2. Venta de 3 meses.

1. Convertir la tabla en un diccionario
2. Modificar el diccionario para incrementar las ventas en un 10%.
3. Hacer un nuevo diccionario pero con las ventas disminuidas un 20%.

Ejercicio propuesto 9: Artículos de tienda

Se tiene la siguiente lista con productos y sus precios. Luego de inspeccionarla se puede ver que probablemente no es la manera más apropiada de almacenar dichos datos.

1. Utilice dos list comprehensions, para extraer las claves y los valores de la lista respectivamente.
2. Cree un Dictionary Comprehension para crear una estructura clave valor. Asegúrese que los valores no sean strings.

```
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"]
```

3. El dueño de la tienda requiere detectar aquellos artículos que cuesten más de $80.000 y aplicarles un descuento de 10%.

Ejercicio propuesto 10: Invertir diccionario

A veces, resulta útil invertir la relación de un diccionario para poder acceder a la llave asociada a un valor en específico. Esto se puede lograr construyendo el diccionario nuevamente y almacenando el valor en el lugar de la llave y viceversa.

¿Cómo se invertiría el siguiente diccionario?

```
colors = {
    "red": "#cc0000",
    "green": "#00cc00",
    "blue": "#0000cc",
}
# Se desea obtener el siguiente resultado
print(colors_inv["#cc0000"]) # red
```

```
'red'
```

Ejercicio propuesto 11: Búsqueda de colores

Se tiene una base de datos de colores:

```
{
  "aliceblue": "#f0f8ff",
  "antiquewhite": "#faebd7",
  "aqua": "#00ffff",
  "aquamarine": "#7fffd4",
  "azure": "#f0ffff",
  "darkorchid": "#9932cc",
  "darkred": "#8b0000",
  "darksalmon": "#e9967a",
  "navajowhite": "#ffdead",
  "navy": "#000080",
  "orchid": "#da70d6",
  "palegoldenrod": "#eee8aa",
  "peachpuff": "#ffdab9",
  "peru": "#cd853f",
  "pink": "#ffc0cb",
  "purple": "#800080",
  "rebeccapurple": "#663399",
  "red": "#ff0000",
  "saddlebrown": "#8b4513",
  "seashell": "#fff5ee",
  "sienna": "#a0522d",
  "silver": "#c0c0c0",
  "skyblue": "#87ceeb",
  "slateblue": "#6a5acd",
  "teal": "#008080",
  "thistle": "#d8bfd8",
  "tomato": "#ff6347",
  "turquoise": "#40e0d0",
  "violet": "#ee82ee",
  "wheat": "#f5deb3",
  "white": "#ffffff",
  "whitesmoke": "#f5f5f5",
  "yellow": "#ffff00",
  "yellowgreen": "#9acd32",
}
```

1. Se pide crear un código capaz de encontrar un color ingresado por el usuario en formato hexadecimal y se muestre en pantalla el nombre del color. En caso de no encontrar el color, aparecerá el texto `"No se encontró el color"`.