Checkpoint 5

¿Qué es un condicional?

- Los condicionales sirven para automatizar el programa, es decir, acatan las órdenes y se ejecutan dependiendo de la orden recibida, el programa funcionará de una manera u otra dependiendo de la condición.
- Expresiones para las condiciones
  - If -> Condiciones que deben cumplirse
    - and -> deben cumplirse todas las condiciones que se ordenen, que se cumpla el nombre de usuario y la contraseña
    - or -> Debe cumplirse por lo menos una de las condiciones
    - and not -> deben cumplirse las condiciones pero siendo falsas,
  - Elif-> Otras condiciones que tendrían que cumplirse,
  - Else -> En caso de no cumplirse las condiciones anteriores
- Operadores condicionales
  - == Igual (string, numbers, listas)
  - != No igual (string, numbers, listas)
  - <> No igual (No usar, anticuado)
  - > Mayor (numbers)
  - >= Mayor o igual (numbers)
  - < Menor (numbers)
  - <= menor o igual (numbers)
- Ejemplo: Para acceder a una web se necesita el nombre y contraseña de usuario. Si ambos son correctos, se podría acceder a la web. Si ninguno de los dos fuera el correcto, no se podría acceder y se ofrecería registrarse. Si alguno de los dos estuviera mal, se notificará que alguno de los datos no es correcto.
  - Usuario y password para entrar en una página.
  - if -> Si ambos son correctos (afirmativo) se le permite la entrada,
    - if nombre == 'Mielma' and contraseña == 'Secreto':
      print('Acceso permitido')
  - elif -> Si ninguno de ellos es correcto (afirmativo), denegar acceso y ofrecer registrarse
    - elif nombre != 'Mielma' and contraseña != 'Secreto':
      print('Sólo para usuarios registrados, ¿Desearías registrarte?')
  - else -> En caso contrario (negativo), Alguno de los datos no es el correcto
    - else:
      print('Alguno de los datos no es el correcto')

- Código del ejemplo,

```
nombre = 'Mielma'
contraseña = 'secreto'


if nombre == 'Mielma' and contraseña == 'Secreto':
  print('Acceso permitido')
elif nombre != 'Mielma' and contraseña != 'Secreto':
   print('Sólo para usuarios registrados, ¿Desearías registrarte?')
else:
  print('Alguno de los datos no es el correcto')
```

¿Cuáles son los diferentes tipos de bucles en Python? ¿Por qué son útiles?

- Los bucles son una herramienta muy importante en la programación, ya que nos permiten ejecutar un bloque de código varias veces seguidas. Se utilizan para repetir varias veces la ejecución de una parte de un programa.
- Son útiles porque en un sólo código repiten la operación las veces que se le haya ordenado
  - Para saber el número consecutivo hay que hacer n+1, hay que repetir esa operación una y otra vez hasta el infinito, pero con el bucle sólo pondremos una vez la operación y el programa lo repetirá tantas veces como sea necesario, hasta alcanzar el objetivo fijado
- Hay dos tipos de bucle
  - for.in.loop -> Bucle tantas veces como elementos haya, aunque no se sepa la cantidad de elementos con los que trabajamos
    - Listas y tuplas: Consejo, si la colección está en plural, usar la variable bloque en singular. Print, tiene que llevar sangría obligatoria,.

      for variable in variables:

      print(variable)
    - Diccionario tenemos que darle dos nombres, key y valor

      for key, valor in diccionarios.items():

      print('key', key)

      print('valor', valor)
  - while -> Bucle infinito, a no ser que le digamos cuando parar
- Ejemplos for.in.loop:
  - string-> Imprime cada carácter y acaba cuando llegue al último. En este caso imprime

    ```
    name = 'eli'
    for character in name:
        print(character)
    ```

    En este caso imprime

    e

    l

    i

    y finaliza al llegar al final

- Iterar rangos -> Cuando sepas que tienes un número determinado de veces que desees recorrer un conjunto de datos específico. Que no desea limitarlo a la cantidad de elementos que hay en la colección.
  - En una lista del 1 al 100 que sólo queramos imprimir parte de la lista

    for num in range(desde, hasta no incluido, intervalo opcional) ->

    Desde el 1 hasta el 10 (no incluido) con un intervalo de 2

```
for num in range(1, 10, 2):
  print(num)
```

- Ejemplo while
  - Mientras la longitud de nums sea mayor que 0, imprimir. Pop “eliminará” del diccionario por lo que irá definiendo el centinela, empieza desde el final de la lista y se detendrá cuando llegue al valor asignado.

```
nums = list(range(1, 100))


while len(nums) > 0:
 print(nums.pop())
```

¿Qué es una lista por comprensión en Python?

- Son una herramienta muy poderosa, es una funcionalidad que nos permite crear listas avanzadas, basadas en otras listas, se pueden usar condicionales.
- La función se escribe en una misma línea, siendo más elegante, teniendo menos código.
- Ejemplo, crear una lista que sume uno a cada elemento de la primera lista.
  - Crear lista

```
numeros = [1, 2, 3, 4] # lista de números

numeros_mas_uno = [] # nueva lista

for n in numeros: # for variable_bloque in nombre_lista

     numeros_mas_uno.append(n + 1) # añadir uno a cada
numero de la lista


print(numeros_mas_uno)
```

  - Lista por comprensión, misma lista creada pero en una sóla línea

```
numeros = [1, 2, 3, 4] # lista de números

comprension = [n + 1 for n in numeros] # lista = bloque
+ 1 for variable_bloque in nombre_lista


print(comprension)
```

¿Qué es un argumento en Python?

- Los argumentos son el nombre de las variables de una función,

```
def sobrenombre(nombre, apellido):
  print(f'{nombre} {apellido}')
```

  - Los argumentos posicionales van en el mismo orden que los hemos puesto. Si hay muchos argumentos, puede crear error, por omisión o confusión en el orden.

```
sobrenombre('Mielma', 'Developer')
```

  - Cuando hay muchos argumentos, llamarlos con el nombre del argumento, como si estuviéramos creando una variable, no es necesario argumentar en el mismo orden si faltara algún argumento, error…

```
sobrenombre(nombre = 'Mielma', apellido = 'Developer')
sobrenombre(apellido = 'Developer', nombre = 'Mielma')
```

  - En caso de no saber cual es el argumento, se pone uno por defecto, por ejemplo para saludar a un invitado

```
def saludo(argumento = 'Invitado'):
  print(f'Hola, {argumento}')
saludo() # Sin argumento, imprime por defecto
saludo('Eli') # Con argumento, omite por defecto
```

¿Qué es una función Lambda en Python?

- Lambda es una herramienta que nos permite empaquetar una función, normalmente más pequeña e introducirla en otras funciones.
- Son móviles y fáciles de usar.
- Toma una variable (varios nombres) y devuelve una cadena formateada.
- Ejemplos:
  - Devuelve la cadena Mielma Developer

```
sobrenombre = lambda nombre, apellido: f'{nombre}
{apellido}'
print(f"""{sobrenombre('Mielma', 'Developer')}
```

  - Devuelve la cadena Hola Mielma Developer

```
def saludo(apodo):
  print(f'Hola {apodo}')

saludo(sobrenombre('Mielma', 'Developer'))
```

¿Qué es un paquete pip?

- pip es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python. Muchos paquetes pueden ser encontrados en el Python Package Index (PyPI) o CheeseShoop.
- pip es un acrónimo recursivo que se puede interpretar como Pip Instalador de Paquetes o Pip Instalador de Python.1
- Se pueden usar los módulos creados por otros desarrolladores.
- Antes de poder usar toda esa biblioteca, tenemos que instalar pip
  - https://bootstrap.pypa.io/get-pip.py
  - Descargar y ejecutar en el terminal -> python get-pip.py
  - Para verificar si está instalada o saber que versión tenemos
    - pip --version
- Una vez instalada, ya podemos usar todas las bibliotecas disponibles en PyPi, que habrá que descargar antes de poder usarlas.
  - https://pypi.org/
  - Buscar paquete que queramos, en este caso Numpy,
  - Abrir terminal y ejecutar -> pip install numpy.
- Una vez descargada la biblioteca que queramos, podemos hacer uso de ella, para eso hay que importarla en el programa que queramos, dos maneras de importar
  - Importarla biblioteca numpy
    - import numpy
  - Importarla biblioteca numpy poniendo un alias np
    - import numpy as np
- Ejemplo
  - Arange -> adelantar un número y el número (cantidad) genera un rango con matriz (Similar a una lista)
    Imprimirá del 0 al 15, ya que el 16 es su tope y el tope no se imprime

```
rango_num = np.arange(16)
print(rango_num)
```

  - Reshape -> Pasa dos números a la vez obtenemos un conjunto anidado con cuatro matrices anidadas dentro de la matriz maestra, cada una con 4 elementos
    Imprimirá lista anidada, 4 lista de 4 elementos

```
rango_num = np.arange(16)
print(rango_num.reshape(4, 4))
```