Planeación de aula.

Identificación

Grado/Grupo: 10	Área/Asignatura: Informática	Fecha : 09-11-23
Docente / C.D.A.: JAIME CABALLERO		
Sede: PRINCIPAL	Periodo Académico: CUATRO	
Eje temático : Comprender la estructura programación por medio de ciclos repetitivos que ayudan en el experiencia de disfrutar el programar acompañado de sus comandos propios de utilizar estativa y global variable		
Tiempo de Ejecución: 20 DIAS		

Aprendizajes

Objetivos de aprendizajes
Desarrolla la habilidad de leer y entender la documentación de las bibliotecas. Esto será crucial a medida que te enfrentes a problemas específicos.
Referentes curriculares (EBC, DBA, Matriz de Referencia, Mallas de Aprendizaje)
Aprendo técnicas para optimizar el rendimiento del código, especialmente si trabajas con conjuntos de datos grandes.
Evidencias de Aprendizajes / Desempeños Esperados
La programación ofrece oportunidades para la creatividad y la innovación. Los estudiantes deben ser alentados a pensar de manera creativa y a buscar soluciones innovadoras a los problemas planteados.
Recursos y materiales
Se utilizara dentro del aula proyector audiovisual, sobre el cual se estará pausando frecuentemente en caso de dudas y profundizar explicaciones.

Momentos de la clase

Inicio /exploración de saberes previos
Se realiza un repaso practico sobre la computadora de todos los comandos dados y estudiados para seguir con el proceso de aprendizaje de la programacion
Contenido / Estructuración

Iterables e iteradores

Para entender al cien por cien los bucles for, y como Python fue diseñado como lenguaje de programación, es muy importante entender los conceptos de `iterables` e `iteradores`. Empecemos con un par de definiciones:

- Los **iterables** son aquellos objetos que como su nombre indica pueden ser iterados, lo que dicho de otra forma es, que puedan ser indexados. Si piensas en un array (o una `list` en Python), podemos indexarlo con `lista[1]` por ejemplo, por lo que sería un iterable.
- Los **iteradores** son objetos que hacen referencia a un elemento, y que tienen un método `next` que permite hacer referencia al siguiente.

Para saber más: Si quieres saber más sobre los iteradores te dejamos este enlace a la documentación oficial.

Ambos son conceptos un tanto abstractos y que pueden ser complicados de entender. Veamos unos ejemplos. Como hemos comentado, los **iterables** son objetos que pueden ser iterados o accedidos con un índice. Algunos ejemplos de iterables en Python son las listas, tuplas, cadenas o diccionarios. Sabiendo esto, lo primero que tenemos que tener claro es que en un `for`, lo que va después del `in` **deberá ser siempre un iterable**.

```
#for <variable> in <iterable>:
#    <Código>
```

Tiene bastante sentido, porque si queremos iterar una variable, esta variable debe ser **iterable**, todo muy lógico. Pero llegados a este punto, tal vez de preguntes ¿pero cómo se yo si algo es iterable o no?. Bien fácil, con la siguiente función `isinstance()` podemos saberlo. No te preocupes si no entiendes muy bien lo que estamos haciendo, fíjate solo en el resultado, `True` significa que es iterable y `False` que no lo es.

```
from collections import Iterable
```

```
lista = [1, 2, 3, 4]
cadena = "Python"
numero = 10
print(isinstance(lista, Iterable))  #True
print(isinstance(cadena, Iterable)) #True
print(isinstance(numero, Iterable)) #False
```

Por lo tanto las listas y las cadenas son iterables, pero `numero`, que es un entero no lo es. Es por eso por lo que no podemos hacer lo siguiente, ya que daría un error. De hecho el error sería `TypeError: int' object is not iterable`.

```
numero = 10
#for i in numero:
#    print(i)
```

Una vez entendidos los **iterables**, veamos los **iteradores**. Para entender los iteradores, es importante conocer la función `iter()` en Python. Dicha función puede ser llamada sobre un objeto que sea iterable, y nos devolverá un iterador como se ve en el siguiente ejemplo.

```
lista = [5, 6, 3, 2]
it = iter(lista)
print(it)       #<list_iterator object at 0x106243828>
print(type(it)) #<class 'list_iterator'>
```

Vemos que al imprimir `it` es un iterador, de la clase `list_iterator`. Esta variable iteradora, hace referencia a la lista original y nos permite acceder a sus elementos con la función `next()`. Cada vez que llamamos a `next()` sobre `it`, nos devuelve el siguiente elemento de la lista original. Por lo tanto, si queremos acceder al elemento 4, tendremos que llamar 4 veces a `next()`. Nótese que el iterador empieza apuntando fuera de la lista, y no hace referencia al primer elemento hasta que no se llama a `next()` por primera vez.

```
lista = [5, 6, 3, 2]
it = iter(lista)
print(next(it))
#     [5, 6, 3, 2]
#      ^
#      |
#     it
print(next(it))
#     [5, 6, 3, 2]
#         ^
#         |
#        it
print(next(it))
#     [5, 6, 3, 2]
#            ^
```

```
#            |
#           it
```

Para saber mas: Existen otros iteradores para diferentes clases:

- `str_iterator` para cadenas
- `list_iterator` para sets.
- `tuple_iterator` para tuplas.
- `set_iterator` para sets.
- `dict_keyiterator` para diccionarios.

Dado que el iterador hace referencia a nuestra lista, si llamamos más veces a `next()` que la longitud de la lista, se nos devolverá un error `StopIteration`. Lamentablemente no existe ninguna opción de volver al elemento anterior.

```
lista = [5, 6]
it = iter(lista)
print(next(it))
print(next(it))
#print(next(it)) # Error! StopIteration
```

Es perfectamente posible tener diferentes iteradores para la misma lista, y serán totalmente independientes. Tan solo dependerán de la lista, como es evidente, pero no entre ellos.

```
lista = [5, 6, 7]
it1 = iter(lista)
it2 = iter(lista)
print(next(it1)) #5
print(next(it1)) #6
print(next(it1)) #7
print(next(it2)) #5
```

Práctica / Transferencia

Vamos a incrementar el desarrollo de un aplicativo que diseñamos en conjunto, esta vez le agregaremos los ciclos WHILE y esperemos la evolución del algoritmo

Descripción de la Evaluación y Valoración/cierre

Desde de sus hogares en los celulares o pc, para los que tengan, de lo contrario llevará escrito el algoritmo en sus libretas interiorizaran el contenido de tal manera que se haga suyo las estructura del mismo.