Planeación de aula.

Identificación

Grado/Grupo: 09	Área/Asignatura: Informática	Fecha : 04-06-23
Docente / C.D.A.: JAIME CABALLERO		
Sede: PRINCIPAL	Periodo Académico: UNO	
Eje temático : Adquirir conocimientos sobre estructuras de datos: Los estudiantes deben entender las estructuras de datos básicas, como matrices, listas enlazadas, pilas y colas. Deben ser capaces de elegir la estructura de datos adecuada para un problema determinado y aplicarla en su código.		
Tiempo de Ejecución: 1 MES		

Aprendizajes

Objetivos de aprendizajes
Comprender los conceptos básicos de la programación: Los estudiantes deben adquirir un conocimiento fundamental de los conceptos de programación, como variables, tipos de datos, estructuras de control (bucles, condicionales), funciones y objetos. Desarrollar habilidades de resolución de problemas: La programación implica resolver problemas utilizando el pensamiento lógico y algorítmico. Los estudiantes deben aprender a descomponer problemas complejos en pasos más pequeños, identificar patrones y diseñar algoritmos para resolverlos.
Referentes curriculares (EBC, DBA, Matriz de Referencia, Mallas de Aprendizaje)
Familiarizarse con un lenguaje de programación: Los estudiantes deben aprender y practicar al menos un lenguaje de programación, como Python, Java o C++. Deben comprender la sintaxis del lenguaje y ser capaces de escribir programas simples.
Evidencias de Aprendizajes / Desempeños Esperados
Fomentar la creatividad y la innovación: 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
Estos objetivos de aprendizaje pueden adaptarse y ampliarse según el nivel y la duración del curso, así como los recursos disponibles
Contenido / Estructuración

Los comando de Python son una base que debes de tener muy sólida, porque son algo que necesitas para poder seguir en tu camino de programador. Como todo debes de entenderlo y saber qué le están diciendo al usuario y lo que la computadora esta leyendo, les voy a dejar la lista de comandos usada en esta vídeo y una descripción breve de para qué sirven.
 	
Print(): este comando sirve para escribir una indicación en el sistema de programación, v.g. cuando pide una introducción de variables numéricas. También es para mostrar resultados.
 	
Cuando se quiere escribir un texto se debe de escribir con comillas, así:
 	
Print(“Buenos días”)
 	
Int(): este comando es utilizado para que Python tome la entrada como un número entero.
 	
Input(): este comando es usado para obtener una respuesta por parte del usuario que está usando el programa. Si no introduce números se marcará un error.
 	
float(): interpreta la entrada como un número decimal. Si no introduce números se marcará un error.
 	
While: es un ciclo que tiene una repetición dominada por la elección del usuario, esta se repite cada que el usuario cumpla con la condición dictada por el ciclo while.
 	
Por si esta breve descripción no es suficiente, les adjunto un vídeo que yo hice, con un ejemplo simple para que puedan, ver lo que hace cada comando en acción.
 	
from __future__ import annotations
from abc import ABC, abstractmethod
 	
class Command(ABC):
  """
  The Command interface declares a method for executing a command.

```
    """

    @abstractmethod
    def execute(self) -> None:
        pass


class SimpleCommand(Command):
    """
    Some commands can implement simple operations on their own.
    """

    def __init__(self, payload: str) -> None:
        self._payload = payload

    def execute(self) -> None:
        print(f"SimpleCommand: See, I can do simple things like printing"
              f"({self._payload})")


class ComplexCommand(Command):
    """
    However, some commands can delegate more complex operations to other
    objects, called "receivers."
    """

    def __init__(self, receiver: Receiver, a: str, b: str) -> None:
        """
        Complex commands can accept one or several receiver objects along with
        any context data via the constructor.
        """

        self._receiver = receiver
        self._a = a
        self._b = b

    def execute(self) -> None:
        """
        Commands can delegate to any methods of a receiver.
        """

        print("ComplexCommand: Complex stuff should be done by a receiver object", end="")
        self._receiver.do_something(self._a)
        self._receiver.do_something_else(self._b)


class Receiver:
    """
```

```
The Receiver classes contain some important business logic. They know how to
perform all kinds of operations, associated with carrying out a request. In
fact, any class may serve as a Receiver.
"""

def do_something(self, a: str) -> None:
    print(f"\nReceiver: Working on ({a}.)", end="")

def do_something_else(self, b: str) -> None:
    print(f"\nReceiver: Also working on ({b}.)", end="")
```

Práctica / Transferencia

Se desarrollar actividades donde el estudiante por medio de la observación de su entorno podrá establecer criterios claros de recursos tecnológicos, su funcionamiento, fallas y requerimientos, explicando al frente de sus compañeros obteniendo expresión oral y manejo del tema.

Descripción de la Evaluación y Valoración/cierre

Todos los estudiantes realizaran escritos investigativos sobre los temas dados, los harán saber a su padres de familia para que en conjunto realicen evaluaciones desde sus hogares.