Fundamentos de Python

Unidad 2. Funciones y estructuras de datos

PROJECTOS

1. Calculadora de Edad
2. Sistema de Evaluación

ESTUDIANTE

Armando Gabriel Jacinto López

INSTITUCION

INFOTEC

FECHA

16/09/2025

1.- Introducción

En el desarrollo de software moderno, **Python** se ha consolidado como uno de los lenguajes de programación más versátiles y de rápido crecimiento. Su sintaxis clara y legible lo hace ideal tanto para principiantes como para desarrolladores experimentados. Para crear programas eficientes, modulares y fáciles de mantener, es fundamental dominar dos conceptos clave: las **funciones** y las **estructuras de datos**. Las funciones nos permiten organizar el código en bloques reutilizables, evitando la repetición y mejorando la legibilidad. Por su parte, las estructuras de datos son las herramientas que nos permiten almacenar y organizar información de manera lógica y eficiente.

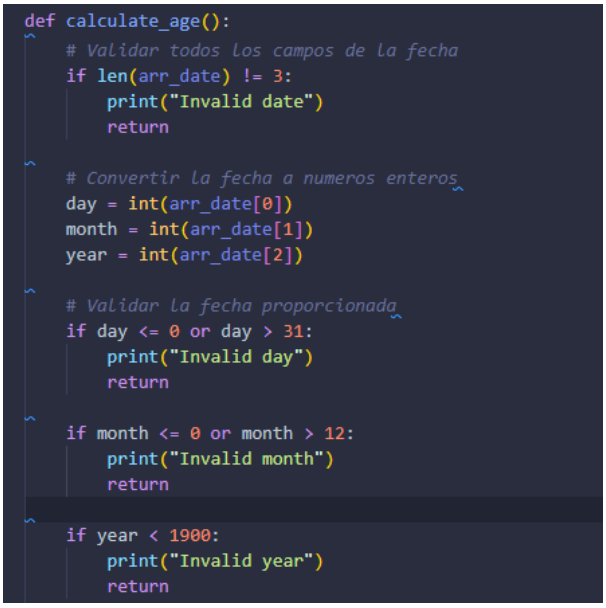
El presente informe analiza la aplicación de estos conceptos a través de dos programas: el primero, un sistema para calcular la edad de un cliente a partir de su fecha de nacimiento, y el segundo, un sistema de gestión académica que almacena información de estudiantes y sus calificaciones. Ambos proyectos demuestran cómo el uso adecuado de funciones y estructuras de datos permite resolver problemas complejos de manera estructurada y optimizada.

2. Desarrollo

2.1. Calculadora de Edad

Este programa tiene como objetivo principal calcular la edad de una persona a partir de su fecha de nacimiento. Para lograrlo, el código hace un uso efectivo de **funciones** y una **estructura de datos** básica.

- **Funciones:** El programa está organizado alrededor de una única función principal, calculate_age(). Esta función encapsula toda la lógica del problema: desde la captura de la entrada del usuario hasta la validación de la fecha y el cálculo de la edad. Al agrupar estas tareas dentro de una función, se logra un código más ordenado y fácil de entender. La función es llamada al final del script, lo que demuestra su papel central en la ejecución del programa.



```
def calculate_age():
    # Validar todos los campos de la fecha
    if len(arr_date) != 3:
        print("Invalid date")
        return

    # Convertir la fecha a numeros enteros
    day = int(arr_date[0])
    month = int(arr_date[1])
    year = int(arr_date[2])

    # Validar la fecha proporcionada
    if day <= 0 or day > 31:
        print("Invalid day")
        return

    if month <= 0 or month > 12:
        print("Invalid month")
        return

    if year < 1900:
        print("Invalid year")
        return
```

Ilustración 1 Función principal y validación de las entradas

- **Estructuras de Datos:** El programa utiliza una **lista (list)** de manera implícita, aunque no se declare directamente con corchetes. Cuando el usuario ingresa la fecha de nacimiento en formato de cadena de texto ("dd-mm-aaaa"), el método split("-") la divide en una lista de cadenas, por ejemplo, ['25', '10', '1995']. Esto permite acceder a cada componente de la fecha (día, mes, año) individualmente y convertirlos a números enteros para su posterior validación y cálculo. Esta es una forma simple pero poderosa de usar una estructura de datos para manipular información.

```python
# Obtener la fecha actual
current_date = datetime.now()

current_day = current_date.day
current_month = current_date.month

# Obtener la edad del usuario
user_age = current_date.year - year
# Aumentar un año, si ya ha pasado el cumpleaños del usuario
if month >= current_month and day >= current_day:
    user_age += 1

# Mostrar la edad
print(user_age)
```

Ilustración 2 Obtención de la edad del usuario

```
$ C:/Users/arman/AppData/Local/Programs/Python/Python313/python.exe
Insert your birthday(dd-mm-aaaa): 27-03-2001
User age: 24 years old
```

Ilustración 3 Ejecución del programa

2.2. Sistema Evaluación

Este programa es más complejo, ya que busca gestionar la información de múltiples estudiantes y sus calificaciones. Para manejar esta complejidad, se apoya fuertemente en **funciones** y **diccionarios**.

- **Funciones:** El programa utiliza dos funciones. La función principal, grades_system(), orquesta todo el proceso: pide el número de estudiantes y materias, y luego entra en bucles para solicitar los datos de cada alumno y sus calificaciones. Además, se crea una función auxiliar llamada verify_inputs(), cuya única tarea es validar que las entradas no estén vacías. El uso de esta función es un excelente ejemplo de **reutilización de código**, ya que se llama múltiples veces para validar diferentes datos de entrada, como el nombre del estudiante, la matrícula, el nombre de la materia y la calificación. Esto evita la duplicación de código y hace que la lógica de la función principal sea más limpia.

```python
# Function prnicipal
def grades_system():
    # Diccionario para almacenar la informacion de los estudiantes
    students = {}
    # Numero de estudiantes y materias
    students_input = input("Insert the students number: ")
    subjects_input = input("Insert the subjects number: ")
    print("\n")
    # Usar la function para verificar las entradas
    num_inputs = verify_inputs(students_input, subjects_input)
    # Comprobar que las entradas son validas
    if not num_inputs:
        return
    # Convertir las entradas a numeros enteros
    students_num = int(students_input)
    subjects_num = int(subjects_input)
```

Ilustración 4 Función principal, entrada, y validación de datos

```python
# Funcion para verificar los datos del usuario
def verify_inputs(input1, input2):
    if not input1.strip() or not input2.strip():
        print("Please Insert Valid Inputs")
        return False
    else:
        return True
```

Ilustración 5 Función para verificar los datos de entrada

- **Estructuras de Datos:** La columna vertebral de este programa es el uso de **diccionarios anidados**. El programa utiliza un diccionario principal llamado students, donde la **clave** es la matrícula del estudiante y el **valor** es otro diccionario. Este diccionario anidado contiene el nombre del estudiante y, a su vez, otro diccionario para almacenar sus materias. Cada materia es una clave, y su valor es un diccionario con la calificación (grade) y el estado (state). Esta estructura jerárquica es ideal para organizar datos complejos y relacionados de forma lógica. Permite un acceso rápido a la información de un estudiante usando su matrícula como clave, lo que es mucho más eficiente que buscar en una lista de estudiantes.

```python
for i in range(students_num):
    # Nombres de los estudiantes y matricula
    student_name = input("Insert student name: ")
    student_tuition = input("Insert student tuition(###): ")
    print("\n")
    # Verificar las entradas
    student_inputs = verify_inputs(student_name, student_tuition)
    # Comprobar que las entradas son validas
    if not student_inputs:
        return
    # Crear una nueva entrada en el diccionario(la llave principal sera la matricula)
    students[student_tuition] = {
        "name": student_name,
        "subjects": {}
    }
```

Ilustración 6 Ciclo for para añadir información sobre los estudiantes

```python
# Ciclo for aninado para pedir informarion sobre las materias y calificaciones
for j in range(subjects_num):
    # Nombre de la materia y calificacion
    subject_name = input("Insert subject name: ")
    subject_input = input("Insert subject grade: ")
    print("\n")
    # Verificar las entradas
    subject_inputs = verify_inputs(subject_name, subject_input)
    # Comprobar si las entradas son validas
    if not subject_inputs:
        return
    # Convertir la calification a un numero flotante
    subject_grade = float(subject_input)
    # Determinar si el estudiante aprobo o reprobo la materia
    if subject_grade >= 6:
        subject_state = "Pass"
    else:
        subject_state = "Fail"
    # Crear una nueva entrada en el diccionario para cada materia
    students[student_tuition]["subjects"][subject_name] = {"grade": subject_grade, "state": subject_state}
```

Ilustración 7 Ciclo for para añadir información sobre las materias

```
Insert student name: armando
Insert student tuition(###): 345


Insert subject name: arts
Insert subject grade: 7


Insert subject name: history
Insert subject grade: 4


Insert student name: gabriel
Insert student tuition(###): 678


Insert subject name: chemistry
Insert subject grade: 4


Insert subject name: physics
Insert subject grade: 9

{'345': {'name': 'armando', 'subjects': {'arts': {'grade': 7.0, 'state': 'Pass'}, 'history':
 {'grade': 4.0, 'state': 'Fail'}}}, '678': {'name': 'gabriel', 'subjects': {'chemistry': {'g
rade': 4.0, 'state': 'Fail'}, 'physics': {'grade': 9.0, 'state': 'Pass'}}}}
```

Ilustración 8 Ejecución del programa, y diccionario final

3. Conclusiones

Los dos programas demuestran de manera práctica cómo las **funciones** y las **estructuras de datos** son pilares fundamentales de la programación en Python. Las funciones, al modularizar la lógica, permiten que el código sea más legible, reutilizable y fácil de mantener, como se vio con la función de validación en el segundo programa.

Por otro lado, la elección de la estructura de datos adecuada es crucial para la eficiencia. El primer programa utiliza una **lista** de manera sencilla para procesar una cadena, mientras que el segundo programa aprovecha el poder de los **diccionarios anidados** para gestionar una base de datos de estudiantes de forma eficiente. En conjunto, ambos proyectos son una clara demostración de que un sólido entendimiento de estos conceptos es esencial para desarrollar programas robustos y bien diseñados.