



UEA  
UNIVERSIDAD  
ESTATAL AMAZÓNICA





**UEA**  
UNIVERSIDAD  
ESTATAL AMAZÓNICA

# **ASIGNATURA**

# **PROGRAMACIÓN ORIENTADA A**

# **OBJETOS**



*Transformamos el mundo desde la Amazonía*

**Ing. Edwin Gustavo Fernández Sánchez, Mgs.**

DOCENTE - PERSONAL ACADÉMICO NO TITULAR OCASIONAL

DIRECTOR DE GESTIÓN DE TECNOLOGÍAS INFORMACIÓN Y COMUNICACIÓN

**UNIVERSIDAD ESTATAL AMAZÓNICA**





**UEA**  
UNIVERSIDAD  
ESTATAL AMAZÓNICA

**SEMANA**

**5**

**DESARROLLO DE LA SEMANA 5: DEL LUN. 06 AL DOM. 12 DE ENERO/2025**

**Resultado de aprendizaje: Aplicar programas que hacen uso de relación de herencia, interfaces y clases abstractas con métodos y variables polimórficas**

## **CONTENIDOS**

### **UNIDAD II: Objetos, clases, Herencia, Polimorfismo**

- Tema 2: Elementos de programación
  - Subtema 2.1: Tipos de datos, Identificadores



**UEA**  
UNIVERSIDAD  
ESTATAL AMAZÓNICA

## **Unidad 2**

**Objetos, clases, Herencia, Polimorfismo.**

### **Tema 1.**

# **Elementos de programación**





**UEA**  
UNIVERSIDAD  
ESTATAL AMAZÓNICA

# **Subtema 1.1: Tipos de datos, Identificadores. Subtema 1.2: Definición de Clase. Subtema 1.3: Definición de Objeto.**

**PROGRAMACIÓN ORIENTADA A  
OBJETOS  
(UEA-L-UFB-030)**



## Definición de Tipos de Datos

Los Tipos de Datos son categorías que nos indican qué clase de información puede almacenar una variable – números, textos, verdadero/falso, entre otros.

Los tipos de datos son una clasificación que especifica qué tipo de valor puede tomar una variable y qué operaciones se pueden realizar sobre ella. Cada tipo de dato en programación tiene un conjunto de características que lo definen, como el tamaño de almacenamiento y cómo se interpreta la información.



## Tipos de Datos Primitivos

Los tipos de datos primitivos son los bloques de construcción básicos de la programación. Son tipos de datos simples proporcionados por el lenguaje de programación. Estos tipos de datos son fundamentales y están diseñados para ser rápidos y eficientes en el manejo de operaciones básicas.

Los datos primitivos son utilizados para crear variables que almacenen información específica, como números para cálculos o caracteres para textos. La correcta elección del tipo de dato primitivo es crucial para la optimización del uso de la memoria y el rendimiento del programa.





## Definición de Tipos de Datos

**Enteros  
(Integers):**  
Números sin  
decimales.

**Ejemplo:**  
5, -3, 42.



**Flotantes  
(Floats):**  
Números con  
decimales.

**Ejemplo:**  
3.14, -0.001.



**Cadenas de Texto  
(Strings):**  
Secuencias de  
caracteres.

**Ejemplo:**  
"Hola", "1234".



**Booleanos  
(Booleans):**  
Valores de  
verdad.

**Ejemplo:**  
Verdadero  
(True), Falso  
(False).



## Tipos de Datos Compuestos

Los tipos de datos compuestos, a diferencia de los primitivos, permiten almacenar y manejar varias piezas de datos, a menudo de diferentes tipos, como una unidad única.

Los datos compuestos permiten al programador construir estructuras más complejas y representar información más rica, como colecciones de objetos, registros, y más. Son fundamentales en la manipulación de datos y estructuras en aplicaciones de software real.



# Definición de Tipos de Datos

**Cadenas de Texto (Strings):**  
Secuencias de caracteres que representan palabras o textos.

**Ejemplo:**  
"Hola Mundo".



**Listas (Lists)/Arreglos (Arrays):**

Colecciones de elementos que pueden ser del mismo tipo o de diferentes tipos.

**Ejemplo en Python:**  
[1, 2, 3] o  
["manzana",  
"banana", 3.14].



**Diccionarios (Dictionaries):**  
Estructuras que almacenan datos en pares clave-valor.

**Ejemplo en Python:**  
{"nombre":  
"Ana", "edad":  
25}.



**Tuplas (Tuples):**  
Similar a las listas, pero inmutables (no pueden ser modificadas una vez creadas).

**Ejemplo en Python:**  
(1, "manzana",  
3.14).



## Identificadores en Programación

Los identificadores son los nombres que asignamos a los elementos en un programa, como variables, clases, funciones y objetos. Funcionan como etiquetas que nos permiten referenciar y manipular estos elementos durante la ejecución del programa.

Los identificadores son esenciales para la claridad y comprensión del código. Un buen identificador describe el propósito o la naturaleza del elemento que representa. Por ejemplo, un identificador como **numeroDeEstudiantes** es más descriptivo y útil que uno como **n** o **num**.



# Reglas para Identificadores

Generalmente, los identificadores deben comenzar con una letra o un guion bajo (\_), seguidos de letras, dígitos o guiones bajos.

Los lenguajes de programación pueden tener reglas específicas y convenciones para los identificadores, como evitar el uso de palabras reservadas o seguir ciertos estilos de nomenclatura (camelCase, snake\_case, etc.).

python

Copy code

```
# Ejemplo de Convenciones de Nomenclatura en Python

# Clases: CamelCase (cada palabra comienza con mayúscula)
class VehiculoElectrico:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def mostrar_informacion(self):
        return f"Vehículo Eléctrico: {self.marca} {self.modelo}"

# Funciones y variables: snake_case (palabras separadas por guiones bajos)
def calcular_distancia(rendimiento, energia):
    return rendimiento * energia

# Constantes: MAYUSCULAS con guiones bajos
MAX_ENERGIA = 100

# Instancia de la clase
mi_tesla = VehiculoElectrico("Tesla", "Model 3")

# Llamada a función
distancia = calcular_distancia(3.5, MAX_ENERGIA)

print(mi_tesla.mostrar_informacion())
print(f"Distancia máxima posible: {distancia} km")
```



## Reglas para Nombrar Identificadores

- Los identificadores en Python deben comenzar con una letra (A-Z/a-z) o un guion bajo (\_).
- Los siguientes caracteres pueden ser letras, dígitos (0-9) o guiones bajos.
- Los identificadores no pueden ser palabras reservadas en Python, como if, for, class, etc.
- En Python, se utiliza comúnmente el estilo **snake\_case** para nombrar variables y funciones.
- Para las clases, se sigue la convención **CamelCase**, donde la primera letra de cada palabra es mayúscula.
- Los identificadores deben ser descriptivos para reflejar su propósito o lo que representan, como **longitud\_lista** o **calcular\_area**.






**UEA**  
UNIVERSIDAD  
ESTATAL AMAZÓNICA

# Clave para la Claridad en el Código

Los identificadores descriptivos son esenciales para crear un código claro y comprensible.

Un buen nombre de identificador explica su propósito o función, facilitando la lectura y comprensión del código sin necesidad de comentarios adicionales.

python

 Copy code

```
# Ejemplo de Buenos Identificadores en Python

# Buen identificador para una variable que almacena la cantidad de estu
cantidad_estudiantes = 25

# Buen identificador para una función que crea un nuevo usuario
def crear_usuario(nombre, edad):
    nuevo_usuario = {'nombre': nombre, 'edad': edad}
    return nuevo_usuario

# Buen identificador para una variable que almacena el precio total de
precio_total = 99.99

# Uso de los identificadores en un contexto de código
print(f"Cantidad de estudiantes: {cantidad_estudiantes}")
usuario = crear_usuario("Laura", 30)
print(f"Usuario creado: {usuario['nombre']} con edad {usuario['edad']}")
print(f"Precio total de la compra: {precio_total}")
```



**UEA**  
UNIVERSIDAD  
ESTATAL AMAZÓNICA


# Impacto de la Elección de Identificadores

Los buenos identificadores hacen que el código sea autoexplicativo, mientras que los malos pueden llevar a confusión y errores, especialmente en proyectos grandes o colaborativos.

En este código, `cnt`, `funcion1`, y `p` son ejemplos de identificadores poco claros.

Estos nombres no proporcionan información suficiente sobre el propósito de la variable o la función, lo que puede llevar a confusión y dificultar la comprensión y el mantenimiento del código.

python

 Copy code

```
# Ejemplo de Identificadores Poco Claros en Python

# Identificador ambiguo para una variable
cnt = 30

# Identificador poco descriptivo para una función
def funcion1(param1, param2):
    resultado = param1 + param2
    return resultado

# Identificador vago para una variable
p = 75.0

# Uso de los identificadores en un contexto de código
print(f"Valor de cnt: {cnt}")
resultado_funcion = funcion1(10, 20)
print(f"Resultado de funcion1: {resultado_funcion}")
print(f"Valor de p: {p}")
```



## Resumen y Mejores Prácticas

- Hemos explorado los 'Tipos de Datos' y los 'Identificadores', dos fundamentos esenciales de la programación.
- Los tipos de datos determinan la naturaleza de la información que podemos procesar, y los identificadores nos permiten nombrar y referenciar diferentes estructuras de programación.
- Seleccionar el tipo de dato correcto es crucial para la eficiencia y la correcta ejecución de un programa.
- Utilizar identificadores claros y descriptivos es fundamental para mantener la legibilidad y comprensibilidad del código.



## Resumen y Mejores Prácticas

- Elige tipos de datos apropiados para tus variables, basándote en lo que necesitas almacenar y procesar.
- Nombra tus variables, funciones, clases y objetos de manera que reflejen su propósito y uso.
- Sigue las convenciones y estándares de nomenclatura de tu lenguaje de programación para mantener la coherencia y claridad.
- **Conclusión:**
  - Entender y aplicar correctamente los tipos de datos e identificadores no solo mejora la calidad de tus programas, sino que también facilita la colaboración y el mantenimiento a largo plazo.
  - Estos conceptos forman la base para desarrollar habilidades de programación más avanzadas y construir aplicaciones más sofisticadas.



UEA  
UNIVERSIDAD  
ESTATAL AMAZÓNICA