Curso	FUNDAMENTOS DE PROGRAMACIÓN CON PYTHON
Tema	MANEJO DE CADENAS Y PROGRAMACION ORIENTADA A OBJETOS
Docente	Mag. Eric Gustavo Coronel Castillo

CADENAS

DEFINICIÓN

Las cadenas en Python (también conocidas como strings) son secuencias de caracteres y se utilizan para representar texto. Aquí tienes algunas formas de definir cadenas en Python.

Comillas dobles o simples

Puedes crear una cadena utilizando comillas dobles (") o comillas simples (').

Por ejemplo:

```
cadena1 = "Esto es una cadena con comillas dobles"
cadena2 = 'Esto es otra cadena con comillas simples'
print(cadena1)
print(cadena2)
```

Cadenas multilínea

Si necesitas incluir varias líneas de texto en una sola cadena, puedes usar comillas triples (''' o """).

Por ejemplo:

```
parrafo = '''Esto es un ejemplo de una cadena multilínea.
Puede abarcar varias líneas sin necesidad de concatenar.'''
print(parrafo)
```

A continuación, tienes otro ejemplo:

```
parrafo = """Esto es otro ejemplo de una cadena multilínea.
Puedes comprobar como escribir varias líneas como una cadena."""
print(parrafo)
```

Secuencias de escape

Para incluir comillas dentro de una cadena, debes utilizar secuencias de escape.

Por ejemplo:

```
cadena = "Esto es una comilla doble \" dentro de una cadena"
print( cadena )
```

Raw strings

Puedes declarar cadenas raw usando el prefijo r.

Las cadenas raw ignoran todas las secuencias de escape.

Por ejemplo:

```
cadena = r"Esto es una cadena con \n comillas dentro"
print( cadena )
```

Otro ejemplo:

```
ruta = r'c:\parent\tasks\new'
print(ruta)
```

Formato de cadenas

Para incluir variables en una cadena, puedes usar el operador + para concatenar o el formateo con % o .format().

Ejemplo sin formato:

```
nombre = "Gustavo"
edad = 40
mensaje = "Hola, soy " + nombre + " y tengo " + str(edad) + " años."
print(mensaje)
```

Ejemplo con formato:

```
nombre = "Gustavo"
edad = 40
mensaje = f"Hola, me llamo {nombre} y tengo {edad} años."
print(mensaje)
```

MANIPULACIÓN DE CADENAS

Longitud de una cadena

Puedes obtener la cantidad de caracteres en una cadena utilizando la función `len`.

Ejemplo:

```
cadena = "Hola, mundo"
longitud = len(cadena)
print(f"La longitud de la cadena es: {longitud}")
```

Búsqueda de subcadenas

Puedes buscar si una subcadena está presente en otra utilizando el método find.

Sintaxis:

---

cadena.find(subcadena)

Si la subcadena no se encuentra, devuelve -1.

Ejemplo:

```
frase = "Python es genial, si es muy genial."
posicion = frase.find("genial")
if posicion != -1:
  print(f"La subcadena 'genial' está en la posición {posicion}")
else:
  print("La subcadena no se encontró")
```

Reemplazo de caracteres

Puedes reemplazar una subcadena por otra utilizando el método replace.

Sintaxis:

cadena.replace(subcadena_original, subcadena_nueva)

Ejemplo:

```
mensaje1 = "Este mundo."
mensaje2 = mensaje1.replace("mundo", "amigo")
print(mensaje1)
print(mensaje2)
```

División de cadenas

Puedes dividir una cadena en partes utilizando el método Split.

Sintaxis:

cadena.split(separador)

Por defecto, se divide por espacios en blanco.

Ejemplo 1:

```
direccion = "Calle Los Heroes 230, Lima"
partes = direccion.split(", ")
print(partes)
```

Ejemplo 2:

```
datos = "Manzana,Naranja,Uva,Pera,Mango"
frutas = datos.split(",")
print(frutas)
```

Formato de cadenas

Puedes formatear cadenas utilizando f-strings o el método format.

Sintaxis:

cadena.format()

Ejemplo:

```
nombre = "Gustavo"
edad = 40
mensaje = "Hola, me llamo {} y tengo {} años."
mensaje = mensaje.format(nombre, edad)
print(mensaje)
```

EJERCICIOS

Ejercicio 1

Desarrollar un programa que permita contar el numero de palabras de un párrafo.

Ejercicio 2

Desarrollar un programa que solicita la fecha de nacimiento de una persona en formato `dd/mm/yyyy`, luego debe mostrar los datos por separado.

Ejercicio 3

Desarrollar un programa que permita el ingreso de la nota de los estudiantes del aula en una cadena separadas por coma.

Por ejemplo: 15,16,10,8,14,13,17,14,15

Luego el programa debe mostrar los siguientes datos:

- La cantidad de estudiantes
- La nota mayor
- La nota menor
- La nota que más se repita
- La media
- La mediana

Ejercicio 4

Crea un programa python que lea una cadena de caracteres y muestre la siguiente información:

- La primera letra de cada palabra. Por ejemplo, si recibe `Universal Serial Bus` debe devolver USB.
- Dicha cadena con la primera letra de cada palabra en mayúsculas. Por ejemplo, si recibe república argentina debe devolver República Argentina.
- Las palabras que comiencen con la letra A. Por ejemplo, si recibe Antes de ayer debe devolver Antes ayer.

PROGRAMACION ORIENTADA A OBJETOS

La programación orientada a objetos (POO) es un paradigma de programación que se basa en la idea de organizar el código en bloques llamados “objetos”. Estos objetos son entidades que pueden tener atributos y comportamientos, y se comunican entre sí a través de mensajes. En POO, el énfasis está en la interacción entre los objetos y en la reutilización de código. En lugar de pensar en el programa como una serie de instrucciones lineales, en POO se piensa en términos de entidades del mundo real y cómo interactúan entre sí.

CONCEPTOS RELACIONADOS

Clases y Objetos

Una clase define las propiedades y comportamientos de un objeto. Los objetos son instancias de una clase.

Por ejemplo, si creamos una clase llamada Coche, los objetos creados a partir de ella serán coches reales con sus propias características (como color, marca, velocidad, etc.).

Modularidad y Reutilización de Código

La POO permite dividir un problema en partes más pequeñas y manejables. Cada clase representa una parte del sistema.

La reutilización de código es una ventaja clave. Puedes aprovechar soluciones y funcionalidades ya implementadas en otros programas o proyectos.

Abstracción y Encapsulación

La abstracción implica definir una clase abstracta que define un conjunto de propiedades y métodos comunes para varias clases relacionadas.

La encapsulación oculta los detalles internos de una clase y expone solo lo necesario para interactuar con ella.

Herencia

Permite crear jerarquías de clases. Una clase puede heredar propiedades y métodos de otra clase base.

Por ejemplo, una clase Empleado puede heredar de una clase Persona.

Polimorfismo

Permite utilizar objetos de diferentes clases de manera intercambiable.

Es útil para escribir código más genérico y flexible.

APLICACIÓN

Clase básica

```
class Drink:
    def __init__(self, name):
        self.name = name

drink = Drink("Agua")
print(drink.name)
```

Encapsulamiento

```
class Drink:
    def __init__(self, name):
        self.__name = name
    def getDetail(self):
        return "La bebida es: " + self.__name

drink = Drink("Agua")
print(drink.getDetail())
```

Herencia

```
class Drink:
    def __init__(self, name):
        self.__name = name
    def getDetail(self):
        return "La bebida es: " + self.__name

class Beer(Drink):
    def __init__(self, name, brand, alcohol):
        super().__init__(name)
        self.__brand = brand
        self.__alcohol = alcohol

beer1 = Beer("Cerveza Trigo","Cristal",10.0)
print(beer1.getDetail())
```

Sobreescritura

```
class Drink:
    def __init__(self, name):
        self.__name = name
    def getDetail(self):
        return "La bebida es: " + self.__name

class Beer(Drink):
    def __init__(self, name, brand, alcohol):
        super().__init__(name)
        self.__brand = brand
        self.__alcohol = alcohol
    def getDetail(self):
        repo = super().getDetail()
        repo += ", la marca es: " + self.__brand
        repo += ", el alcohol es: " + str(self.__alcohol)
        return repo

beer1 = Beer("Cerveza Trigo","Cristal",10.0)
print(beer1.getDetail())
```