Cómputo

Dr. Cruz González Juan Carlos

15 de octubre de 2024

Índice general

Capítulo 1

Lenguaje binario

1.1. Introducción

El código binario y el código ASCII son esenciales en el campo de la computación. Mientras que el código binario es el lenguaje fundamental que utilizan las computadoras para procesar información, el código ASCII permite la representación de caracteres en texto legible para humanos. Este documento explorará en profundidad ambos sistemas de codificación, sus conversiones, ejemplos y aplicaciones.

1.2. Código Binario

El código binario es un sistema de numeración en base 2 que utiliza solo dos dígitos: 0 y 1. En este sistema, cada dígito se conoce como un **bit**. La representación binaria es fundamental para el funcionamiento de las computadoras, ya que estas operan utilizando señales eléctricas que pueden estar encendidas (1) o apagadas (0).

1.2.1. Conversión de Decimal a Binario

Para convertir un número decimal a binario, se utiliza el método de **división sucesiva**:

1. Divide el número por 2.
2. Anota el residuo (0 o 1).
3. Continúa dividiendo el cociente por 2 hasta que el cociente sea 0.
4. El número binario se forma leyendo los residuos de abajo hacia arriba.

Ejemplo de Conversión

Convertimos el número decimal 13 a binario:

$$\begin{aligned} 13 \div 2 = 6 \quad & \text{Residuo: } 1 \\ 6 \div 2 = 3 \quad & \text{Residuo: } 0 \\ 3 \div 2 = 1 \quad & \text{Residuo: } 1 \\ 1 \div 2 = 0 \quad & \text{Residuo: } 1 \end{aligned}$$

Leyendo los residuos de abajo hacia arriba, obtenemos que 13 en decimal es 1101 en binario.

1.2.2. Conversión de Binario a Decimal

Para convertir un número binario a decimal, se utiliza la suma de potencias de 2:

- Comienza desde el bit menos significativo (a la derecha).
- Multiplica cada bit por 2^n, donde n es la posición del bit (comenzando en 0).

Ejemplo de Conversión

Convertimos 1101 a decimal:

$$1 \cdot 2^3 = 8$$
$$1 \cdot 2^2 = 4$$
$$0 \cdot 2^1 = 0$$
$$1 \cdot 2^0 = 1$$

Sumando todos los valores: $8 + 4 + 0 + 1 = 13$.

1.2.3. Ejemplos Prácticos

- El número decimal 10 en binario es 1010.
- El número decimal 255 en binario es 11111111.
- El número binario 101010 en decimal es 42.

1.3. Código ASCII

El código ASCII (American Standard Code for Information Interchange) es un sistema de codificación que asigna números a caracteres. Utiliza un conjunto de 128 caracteres, que incluye letras, números y símbolos de control.

1.3.1. Representación de Caracteres

Cada carácter en ASCII se representa con un número entero en el rango de 0 a 127. Por ejemplo:

- `A` = 65
- `B` = 66
- `C` = 67
- `a` = 97
- `1` = 49
- `!` = 33

1.3.2. Conversión de ASCII a Binario

Para convertir un carácter ASCII a su representación binaria, se convierte el valor decimal correspondiente a su código ASCII a binario.

Ejemplo de Conversión

El carácter `A` tiene un valor ASCII de 65, que en binario se convierte de la siguiente manera:

$$\begin{aligned} 65 \div 2 &= 32 \quad \text{Residuo: } 1 \\ 32 \div 2 &= 16 \quad \text{Residuo: } 0 \\ 16 \div 2 &= 8 \quad \text{Residuo: } 0 \\ 8 \div 2 &= 4 \quad \text{Residuo: } 0 \\ 4 \div 2 &= 2 \quad \text{Residuo: } 0 \\ 2 \div 2 &= 1 \quad \text{Residuo: } 0 \\ 1 \div 2 &= 0 \quad \text{Residuo: } 1 \end{aligned}$$

Leyendo los residuos, 65 en decimal es 1000001 en binario.

1.3.3. Tabla ASCII

A continuación se muestra una parte de la tabla ASCII:

Decimal	Carácter	Binario
32	(espacio)	00100000
33	!	00100001
65	A	01000001
66	B	01000010
67	C	01000011
97	a	01100001
98	b	01100010
99	c	01100011

1.3.4. Ejemplos Prácticos

- El carácter `Z` tiene un valor ASCII de 90, que es 01011010 en binario.
- El carácter `9` tiene un valor ASCII de 57, que es 00111001 en binario.
- El carácter `?` tiene un valor ASCII de 63, que es 00111111 en binario.

1.4. Aplicaciones Prácticas

El conocimiento del código binario y ASCII tiene aplicaciones en diversos campos, tales como:

- **Programación**: Los programadores utilizan estas codificaciones para manipular texto y datos en sus programas.
- **Redes**: El protocolo de comunicación se basa en la interpretación de datos binarios.
- **Hardware**: Los circuitos digitales operan en base a señales binarias.
- **Compresión de Datos**: Los algoritmos de compresión se apoyan en representaciones binarias para optimizar el espacio.

1.5. Conclusión

El código binario y el código ASCII son conceptos fundamentales en la informática. Comprender cómo funcionan estos sistemas de codificación no solo es esencial para programadores, sino también para cualquier persona interesada en la tecnología. Estos códigos son la base sobre la cual se construyen aplicaciones, redes y sistemas operativos.

Capítulo 2

Entorno y sintaxis básica

2.1. Introducción

Python es un lenguaje de programación de alto nivel, conocido por su simplicidad y legibilidad. Es ampliamente utilizado en diversas áreas, desde el desarrollo web hasta la ciencia de datos. En esta sección aprenderemos los primeros pasos para trabajar con Python, cubriendo su sintaxis básica y cómo ejecutar programas.

2.2. Sintaxis Básica

2.2.1. Comentarios

En Python, los comentarios se usan para agregar anotaciones dentro del código que no son ejecutadas. Se pueden agregar comentarios de una sola línea usando el símbolo #:

```
# Este es un comentario de una sola línea
print("Hola, Mundo!")  # Esto imprime un mensaje
```

Listing 2.1: Código con comentarios

Los comentarios de varias líneas pueden hacerse con comillas triples (''' o """):

```
'''
Este es un comentario
de varias líneas
'''
```

Listing 2.2: Comentarios de varias líneas

2.2.2. Variables y Tipos de Datos

En Python, para la creación de una variable se sigue la siguiente sintaxis:

```
Nombre_variable = Valor_variable #El nombre lo pueden elegir
```

Listing 2.3: Creación de variables

El símbolo = **no representa la igualdad, si no la asignación del valor.** a una variable no es necesario declarar explícitamente el tipo de una variable. Simplemente asignamos valores a las variables y el lenguaje determina automáticamente el tipo de dato:

```python
x = 5           # Entero
y = 3.14        # Flotante
nombre = "Juan" # Cadena de texto
es_activo = True # Booleano
```

Listing 2.4: Asignación de variables

2.2.3. Impresión en consola

La función `print()` se usa para mostrar información en la consola:

```python
print("Hola, Mundo!")
print("El valor de x es:", x)
```

Listing 2.5: Uso de print()

Salida esperada:

```
Hola, Mundo!
El valor de x es: 5
```

2.2.4. Operadores básicos

Python soporta los operadores comunes como suma, resta, multiplicación y división:

```python
a = 10
b = 3
suma = a + b          # Suma
resta = a - b         # Resta
producto = a * b      # Multiplicación
division = a / b      # División (flotante)
division_entera = a // b # División entera
modulo = a % b        # Residuo
potencia = a ** b     # Exponenciacion
```

Listing 2.6: Operaciones básicas

2.3. Ejecución de Programas en Python

2.3.1. Ejecución en el intérprete interactivo

El intérprete interactivo de Python permite escribir y ejecutar código línea por línea. Para acceder al intérprete, simplemente abre una terminal o consola y escribe `python` o `python3`, dependiendo de tu instalación. Una vez dentro, puedes escribir código y ver el resultado inmediatamente:

```
$ python3
>>> print("Hola, Mundo!")
Hola, Mundo!
```

2.3.2. Ejecución de scripts

Para ejecutar un archivo de Python (`.py`), puedes escribir tu código en un editor de texto, guardar el archivo, y luego ejecutarlo desde la línea de comandos:

- Escribe el siguiente código en un archivo llamado `hola.py`:

```python
    print("Hola, este es mi primer programa en Python")

```

- Luego, en la terminal, navega al directorio donde guardaste el archivo y ejecuta el siguiente comando:

```
$ python3 hola.py
```

Salida esperada:

```
Hola, este es mi primer programa en Python
```

2.3.3. Uso de entornos como IDLE o VSCode o Spyder

Además de la terminal, puedes usar entornos de desarrollo integrados (IDE) como `IDLE`, que viene preinstalado con Python, o editores como `Visual Studio Code` para escribir y ejecutar tu código de manera más cómoda al igual que Spyder.

- En `IDLE`, abre el editor de texto, escribe tu código y selecciona `Run >Run Module` para ejecutarlo.
- En `VSCode`, puedes instalar la extensión de Python para ejecutar scripts directamente desde el editor.

2.4. Ejercicios

2.4.1. Ejercicio 1: Suma, Resta, Multiplicación y División

Escribe un programa que pida al usuario dos números y luego imprima los resultados de las operaciones básicas (suma, resta, multiplicación y división) entre ellos.

```python
# Solicitar al usuario dos números
num1 = float(input("Introduce el primer número: "))
num2 = float(input("Introduce el segundo número: "))

# Realizar operaciones básicas
suma = num1 + num2
resta = num1 - num2
multiplicacion = num1 * num2
division = num1 / num2

# Mostrar los resultados
print(f"Suma: {suma}")
print(f"Resta: {resta}")
print(f"Multiplicación: {multiplicacion}")
print(f"División: {division}")
```

Listing 2.7: Código para el ejercicio 1

2.4.2. Ejercicio 2: Cálculo de área de un círculo

Escribe un programa que calcule el área de un círculo dado su radio. Usa la fórmula:

$$\text{Área} = \pi \cdot r^2$$

donde $\pi = 3{,}1416$.

```python
import math

# Solicitar el radio del círculo
radio = float(input("Introduce el radio del círculo: "))

```

```python
# Calcular el área
area = math.pi * radio ** 2

# Mostrar el resultado
print(f"El área del círculo es: {area}")
```

Listing 2.8: Código para el ejercicio 2

2.4.3. Ejercicio 3: Operaciones combinadas

Escribe un programa que pida tres números y realice las siguientes operaciones:

- Suma los tres números.
- Multiplica el primer número por el segundo y luego resta el tercero.
- Divide la suma de los tres números entre 3.

```python
# Solicitar tres números
a = float(input("Introduce el primer número: "))
b = float(input("Introduce el segundo número: "))
c = float(input("Introduce el tercer número: "))

# Operaciones
suma_total = a + b + c
operacion_2 = (a * b) - c
media = suma_total / 3

# Mostrar resultados
print(f"Suma de los tres números: {suma_total}")
print(f"Resultado de (a * b) - c: {operacion_2}")
print(f"Promedio de los tres números: {media}")
```

Listing 2.9: Código para el ejercicio 3

2.4.4. Ejercicio 4: Conversión de grados Celsius a Fahrenheit

Escribe un programa que convierta una temperatura dada en grados Celsius a grados Fahrenheit usando la fórmula:

$$F = \frac{9}{5}C + 32$$

```python
# Solicitar la temperatura en Celsius
celsius = float(input("Introduce la temperatura en grados Celsius: "))

# Convertir a Fahrenheit
fahrenheit = (celsius * 9/5) + 32

# Mostrar el resultado
print(f"La temperatura en Fahrenheit es: {fahrenheit}")
```

Listing 2.10: Código para el ejercicio 4

2.4.5. Ejercicio 5: Cálculo del resto

Escribe un programa que pida al usuario dos números enteros y muestre el resto de la división entre ellos (usando el operador módulo %).

```python
# Solicitar dos números enteros
a = int(input("Introduce el primer número entero: "))
b = int(input("Introduce el segundo número entero: "))

# Calcular el resto
resto = a % b

# Mostrar el resultado
print(f"El resto de {a} dividido por {b} es: {resto}")
```

Listing 2.11: Código para el ejercicio 5

2.4.6. Ejercicio 6: Potencia de un número

Escribe un programa que pida al usuario una base y un exponente, y luego calcule el resultado de elevar la base al exponente.

```python
# Solicitar base y exponente
base = float(input("Introduce la base: "))
exponente = float(input("Introduce el exponente: "))

# Calcular la potencia
resultado = base ** exponente

# Mostrar el resultado
print(f"{base} elevado a {exponente} es: {resultado}")
```

Listing 2.12: Código para el ejercicio 6

2.4.7. Ejercicio 7: Conversión de segundos a horas, minutos y segundos

Escribe un programa que convierta una cantidad de segundos en horas, minutos y segundos.

```python
# Solicitar los segundos
segundos = int(input("Introduce la cantidad de segundos: "))

# Calcular horas, minutos y segundos
horas = segundos // 3600
minutos = (segundos % 3600) // 60
segundos_restantes = segundos % 60

# Mostrar el resultado
print(f"{segundos} segundos son: {horas} horas, {minutos} minutos y {segundos_restantes}
    segundos.")
```

Listing 2.13: Código para el ejercicio 7

2.5. Conclusión

Con estos conceptos básicos, ya puedes empezar a escribir y ejecutar programas simples en

Capítulo 3

Estructuras de control

3.1. Introducción

Las estructuras de control en Python permiten dirigir el flujo de ejecución del programa, tomando decisiones, repitiendo acciones o alterando el curso normal según ciertas condiciones. A continuación, exploraremos las estructuras condicionales, los bucles y las sentencias de control de flujo.

3.2. Condicionales

Las estructuras condicionales en Python permiten que se ejecute un bloque de código si se cumple una condición. Se utilizan las palabras clave `if`, `elif` y `else`.

3.2.1. Sintaxis de `if`

El bloque `if` se ejecuta si la condición es verdadera:

```
x = 10
if x > 5:
    print("x es mayor que 5")
```

Listing 3.1: Uso de la sentencia if

Salida esperada:

```
x es mayor que 5
```

3.2.2. Uso de `else`

Si la condición es falsa, se ejecuta el bloque `else`:

```
x = 3
if x > 5:
    print("x es mayor que 5")
else:
    print("x es menor o igual que 5")
```

Listing 3.2: Uso de if-else

Salida esperada:

```
x es menor o igual que 5
```

3.2.3. Uso de `elif`

La palabra clave `elif` permite añadir condiciones adicionales si las anteriores no se cumplen:

```
x = 5
if x > 5:
    print("x es mayor que 5")
elif x == 5:
    print("x es igual a 5")
else:
    print("x es menor que 5")
```

Listing 3.3: Uso de if-elif-else

Salida esperada:

```
x es igual a 5
```

3.3. Bucles

Los bucles permiten repetir un bloque de código varias veces. Python tiene dos tipos principales de bucles: `for` y `while`.

3.3.1. Bucle `for`

El bucle `for` itera sobre los elementos de una secuencia, como una lista o una cadena de caracteres.

```
# Itera sobre una lista
numeros = [1, 2, 3, 4, 5]
for numero in numeros:
    print(numero)
```

Listing 3.4: Uso del bucle for

Salida esperada:

```
1
2
3
4
5
```

También puedes usar `range()` para generar una secuencia de números:

```
# Imprime los números del 0 al 4
for i in range(5):
    print(i)
```

Listing 3.5: Uso de range con for

Salida esperada:

```
0
1
2
3
4
```

3.3.2. Bucle `while`

El bucle `while` repite un bloque de código mientras una condición sea verdadera.

```
x = 0
while x < 5:
    print(x)
    x += 1  # Incrementa x en 1
```

Listing 3.6: Uso del bucle while

Salida esperada:

```
0
1
2
3
4
```

3.4. Sentencias de Control de Flujo

Las sentencias de control de flujo permiten modificar el comportamiento normal de los bucles.

3.4.1. Sentencia `break`

La sentencia `break` se utiliza para salir de un bucle antes de que finalice su ciclo completo:

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

Listing 3.7: Uso de break

Salida esperada:

```
0
1
2
3
4
```

3.4.2. Sentencia `continue`

La sentencia `continue` salta a la siguiente iteración del bucle sin ejecutar el resto del código en esa iteración:

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

Listing 3.8: Uso de continue

Salida esperada:

```
0
1
2
4
```

3.4.3. Sentencia `pass`

La sentencia `pass` no hace nada; se usa como marcador de posición cuando se necesita una sintaxis válida pero no se quiere ejecutar ninguna acción.

```
for i in range(5):
    if i == 3:
        pass  # No se hace nada en esta iteración
    print(i)
```

Listing 3.9: Uso de pass

Salida esperada:

```
0
1
2
3
4
```

3.5. Ejercicios.

Ejercicio 1: Número par o impar

Escribe un programa que solicite al usuario un número entero y determine si es par o impar utilizando una estructura condicional `if-else`.

```
numero = int(input("Ingresa un número entero: "))
if numero % 2 == 0:
    print("El número es par")
else:
    print("El número es impar")
```

Ejercicio 2: Clasificación de edades

Crea un programa que solicite una edad y determine si la persona es un niño (menos de 12 años), adolescente (entre 12 y 18 años), adulto (entre 18 y 60 años) o adulto mayor (más de 60 años) usando `if-elif-else`.

```
edad = int(input("Ingresa tu edad: "))
if edad < 12:
    print("Eres un niño")
elif 12 <= edad < 18:
    print("Eres un adolescente")
elif 18 <= edad < 60:
    print("Eres un adulto")
else:
    print("Eres un adulto mayor")
```

Ejercicio 3: Factorial de un número

Escribe un programa que calcule el factorial de un número ingresado por el usuario utilizando un bucle `for`.

```
numero = int(input("Ingresa un número: "))
factorial = 1
for i in range(1, numero + 1):
    factorial *= i
print("El factorial de", numero, "es", factorial)
```

Ejercicio 4: Suma de números pares

Crea un programa que sume todos los números pares del 1 al 100 usando un bucle `for`.

```
suma = 0
for i in range(1, 101):
    if i % 2 == 0:
        suma += i
print("La suma de los números pares del 1 al 100 es", suma)
```

Ejercicio 5: Contador de vocales

Escribe un programa que cuente cuántas vocales tiene una palabra ingresada por el usuario usando un bucle `for` y una condición `if`.

```
palabra = input("Ingresa una palabra: ").lower()
vocales = "aeiou"
contador = 0
for letra in palabra:
    if letra in vocales:
        contador += 1
print("La palabra tiene", contador, "vocales")
```

Ejercicio 6: Adivina el número

Escribe un programa que simule un juego de adivinar un número. El programa debe generar un número aleatorio entre 1 y 10, y permitir al usuario adivinarlo usando un bucle `while` hasta que lo adivine.

```
import random

numero_secreto = random.randint(1, 10)
adivina = 0

while adivina != numero_secreto:
    adivina = int(input("Adivina el número (1-10): "))
    if adivina < numero_secreto:
        print("El número es mayor")
    elif adivina > numero_secreto:
        print("El número es menor")

print("¡Adivinaste!")
```

Ejercicio 7: Números primos

Escribe un programa que imprima todos los números primos entre 1 y 50 usando un bucle `for` y una condición `if`.

```
for num in range(2, 51):
    primo = True
    for i in range(2, num):
        if num % i == 0:
            primo = False
            break
    if primo:
        print(num)
```

Ejercicio 8: Tabla de multiplicar

Escribe un programa que muestre la tabla de multiplicar de un número ingresado por el usuario utilizando un bucle `for`.

```python
numero = int(input("Ingresa un número: "))
for i in range(1, 11):
    print(numero, "x", i, "=", numero * i)
```

Ejercicio 9: Suma de dígitos

Crea un programa que solicite un número entero y calcule la suma de sus dígitos utilizando un bucle `while`.

```python
numero = int(input("Ingresa un número: "))
suma = 0
while numero > 0:
    digito = numero % 10
    suma += digito
    numero = numero // 10
print("La suma de los dígitos es:", suma)
```

Ejercicio 10: Números divisibles por 3 y 5

Escribe un programa que imprima todos los números del 1 al 100 que sean divisibles por 3 y por 5 usando un bucle `for`.

```python
for i in range(1, 101):
    if i % 3 == 0 and i % 5 == 0:
        print(i)
```

3.6. Conclusión

Las estructuras de control son esenciales para cualquier lenguaje de programación, ya que permiten a los programas tomar decisiones y ejecutar bloques de código de manera repetitiva. La combinación de condicionales, bucles y sentencias de control de flujo proporciona una gran flexibilidad para manejar diversos casos en la lógica de un programa.