Resumen Certamen 1

INF129

Autor: Eduardo Pino H.

26 de abril de 2025

Índice

1. Laboratorio 1A: Programas Secuenciales en Python

Operadores Aritméticos

Operador	Descripción
+	Suma entre dos números
-	Resta entre dos números
*	Multiplicación
/	División estándar (con decimales)
//	División entera (sin decimales)
%	Módulo (resto de la división)
**	Potencia (exponente)

1.1. Precedencia de Operadores (de mayor a menor)

1. Paréntesis `()`
2. Potencia `**`
3. Multiplicación, División, División entera, Módulo `*, /, //, %`
4. Suma y Resta `+, -`

Ejemplos

```
resultado1 = 5 + 2 * 3          # resultado1 = 11
resultado2 = (5 + 2) * 3        # resultado2 = 21
resultado3 = 10 // 3            # resultado3 = 3
resultado4 = 10 % 3             # resultado4 = 1
resultado5 = 2 ** 3             # resultado5 = 8
```

1.2. Declaración de Variables

```
a = 10
b = 3
c = a / b
print("Resultado:", c)          # Resultado: 3.33...
```

1.3. Operaciones con Variables

```
a = 8
b = 4

suma = a + b
resta = a - b
```

```
multi = a * b
div = a / b

print("Resultados:", suma, resta, multi, div)
# Resultados: 12 4 32 2.0
```

1.4. Conversión de Tipos (Casteo)

```
a = "15"
print(int(a))  # 15 (Convierte string a entero)

b = 567.98
print(str(b))  # "567.98" (Convierte float a string)

c = "34.98"
print(float(c))  # 34.98 (Convierte string a float)

d = 456.6763
print(int(d))  # 456 (Trunca el decimal)
```

Ejemplo Comparativo

```
numero1 = 14.56
numero2 = 6.9

resultado1 = int(numero1 * numero2)
resultado2 = int(numero1) * int(numero2)

print(resultado1)   # 100
print(resultado2)   # 84
```

1.5. Importar una biblioteca

```
import math
import random
```

Ejemplos con `math`

```
import math

raiz = math.sqrt(16)
print("Raiz cuadrada:", raiz)           # Raiz cuadrada: 4.0
```

```
potencia = math.pow(2, 4)
print("2 elevado a 4:", potencia)       # 2 elevado a 4: 16.0

redondeo_arriba = math.ceil(5.2)
print("Redondeo hacia arriba:", redondeo_arriba)  # 6
```

Ejemplos con `random`

```
import random

numero_aleatorio = random.randint(1, 10)
print("Numero aleatorio:", numero_aleatorio)  # Ejemplo: 7
```

1.6. Funciones Matemáticas Nativas de Python

```
print(abs(-42))        # 42 (valor absoluto)
print(round(3.14159))  # 3 (redondeo)
print(round(3.14159, 2))  # 3.14 (redondeo a 2 decimales)
```

2. Laboratorio 1B: Programas Secuenciales en Python

2.1. Secuencialidad

En programación, la **secuencialidad** significa que las instrucciones se ejecutan una tras otra en el orden en que aparecen, de arriba hacia abajo.

```
x = 10             # Paso 1
y = x + 5          # Paso 2
print(y)           # Paso 3 (Resultado: 15)
```

Importancia del orden: si usamos una variable antes de asignarle un valor, Python mostrará un error.

```
y = x + 5          # Error: x no esta definida aun
x = 10
print(y)
```

Corrección:

```
x = 10
y = x + 5
print(y)           # Resultado: 15
```

2.2. Entrada y Salida de Datos

Salida de datos: usamos la función `print()` para mostrar información al usuario.

```
nombre = "Benjamin"
print("Hola", nombre)   # Hola Benjamin
```

Entrada de datos: usamos `input()` para recibir datos del usuario. Por defecto, lo ingresado es un string.

```
edad = input("Ingresa tu edad: ")
print("Tu edad es:", edad)
```

Conversión de entrada:

Para operar con números, debemos convertir el valor ingresado (casteo):

```
numero = int(input("Ingresa un numero: "))
print("El doble es: ", numero * 2)
```

Para decimales:

```
decimal = float(input("Ingresa un numero decimal: "))
print("La mitad es:", decimal / 2)
```

3. Laboratorio 2: Condicionales

3.1. Operadores

Operadores de Comparación
Permiten comparar dos valores. Retornan `True` o `False`.

Operador	Descripción	Ejemplo
==	Igual que	`5 == 5` → True
!=	Diferente que	`4 != 2` → True
<	Menor que	`3 <8` → True
<=	Menor o igual que	`6 <= 6` → True
>	Mayor que	`10 >5` → True
>=	Mayor o igual que	`7 >= 8` → False

Operadores Lógicos
Permiten combinar condiciones.

- `and`: Verdadero si ambas condiciones son verdaderas.
- `or`: Verdadero si al menos una condición es verdadera.
- `not`: Invierte el valor de la condición.

Ejemplos:

```
print(True and False)     # False
print(True or False)      # True
print(not True)           # False
```

3.2. Condicionales

Los condicionales permiten que un programa tome decisiones basadas en condiciones.

Condicional if

Ejecuta un bloque si la condición es verdadera.

```
edad = int(input("Ingresa tu edad: "))
if edad >= 18:
    print("Eres mayor de edad")
```

Condicional if-else

Ejecuta un bloque si la condición es verdadera, y otro si es falsa.

```
temperatura = int(input("Temperatura actual: "))
if temperatura > 20:
    print("Hace calor")
else:
    print("Hace frio")
```

Condicional if-elif-else

Evalúa múltiples condiciones en orden.

```
nota = int(input("Ingresa tu nota: "))
if nota >= 90:
    print("Excelente")
elif nota >= 70:
    print("Bueno")
elif nota >= 55:
    print("Regular")
else:
    print("Malo")
```

Nota: Python evalúa de arriba hacia abajo y se detiene en la primera condición verdadera (cortocircuito).

Condicionales Anidados

Permiten decisiones más complejas:

```
dia = "Lunes"
hora = 10

if dia == "Lunes" or dia == "Martes":
    if hora < 13:
        print("Es hora de clases")  # Se ejecutara este print
    else:
        print("Es hora de almuerzo")
else:
    print("No es lunes ni martes")
```

4. Laboratorio 3: Ciclos While

4.1. Entendiendo el Ciclo While

El ciclo `while` permite repetir un bloque de código mientras una condición sea verdadera.

```
n = 1
while n <= 10:
    print(n)
    n += 1
```

Conceptos importantes:

- La condición se evalúa antes de cada iteración.
- Si la condición es falsa al inicio, el ciclo no se ejecuta.
- Se debe actualizar la variable para evitar ciclos infinitos.

Ejemplo con entrada de usuario:

```
maximo = int(input("Ingrese el numero maximo: "))
n = 1

if maximo < 1:
    print("Numero invalido, no se puede contar.")
else:
    while n <= maximo:
        print(n)
        n += 1
```

Ciclos Infinitos

Ocurren cuando la condición siempre es verdadera.

```
n = 1
while n > 0:
    print(n)
    n += 1
# Ctrl+C para detener manualmente
```

Ciclos Anidados

Es posible colocar un `while` dentro de otro.

```
tabla = 1
maximo = int(input("Ingrese un numero para generar tablas: "))

while tabla <= maximo:
    print("Tabla del", tabla)
    numero = 1
    while numero <= 10:
        print(tabla, "*", numero, "=", tabla * numero)
        numero += 1
    tabla += 1
```

4.2. Patrones Comunes con Ciclos While

Patrón 1: Cantidad fija de repeticiones

Usa un contador para controlar el número de repeticiones.

```
contador = 10
while contador > 0:
    print("Despegue en...", contador)
    contador -= 1
print("Despegue")
```

Patrón 2: Contadores

Se utiliza un contador para acumular datos.

```
calificaciones_totales = 0
num_estudiantes = 5

while num_estudiantes > 0:
    calificacion = int(input("Ingrese la calificacion: "))
    calificaciones_totales += calificacion
    num_estudiantes -= 1
```

```
promedio = calificaciones_totales / 5
print("Promedio de calificaciones:", promedio)
```

Patrón 3: Valor específico como condición

Continuar hasta que se adivine un número.

```
import random

secreto = random.randint(1, 10)
adivinado = False

while not adivinado:
    intento = int(input("Adivina el numero magico (1-10): "))
    if intento == secreto:
        print("Has adivinado el numero magico")
        adivinado = True
    else:
        print("Incorrecto! Intenta de nuevo.")
```

Patrón 4: Uso de flags (variables como condición)

Permite salir del ciclo cuando el usuario lo decide.

```
total = 0
comprando = True

print("Bienvenido a la caja registradora.")
print("Escribe 'fin' para terminar.")

while comprando:
    precio = input("Ingrese el precio del producto (o 'fin'): ")
    if precio.lower() == "fin":
        comprando = False
    else:
        total += float(precio)

print("Total a pagar:", total)
```

5. Laboratorio 4: Strings y Ciclos For

Strings

Un **string** es una secuencia ordenada de caracteres, delimitada entre comillas simples (' ') o dobles (" ").

Acceso a caracteres:

```
texto = "Monty Python"
print(texto[0])   # 'M'
print(texto[3])   # 't'
print(texto[-1])  # 'n' (indice negativo)
```

Substrings y operador de slice:

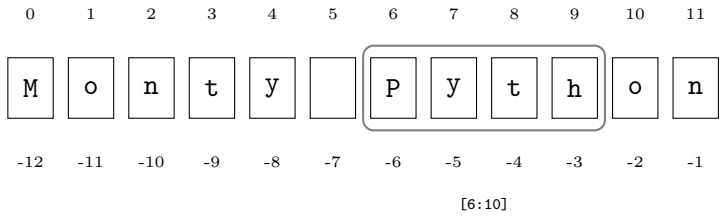

Figura 1: Slice [6:10] cubre los índices 6 al 9, seleccionando "Pyth"

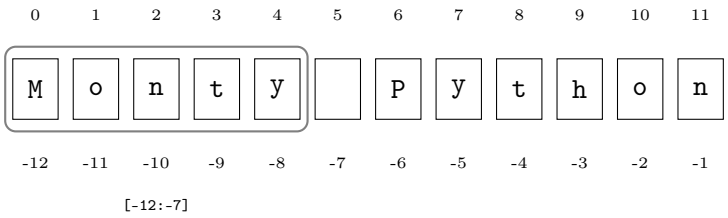

Figura 2: Slice [-12:-7] cubre los índices -12 al -8, seleccionando "Monty"

```
print(texto[6:10])     # 'Pyth'
print(texto[:5])       # 'Monty'
print(texto[6:])       # 'Python'
print(texto[:])        # 'Monty Python'
```

Importante: - El índice final no se incluye en el resultado. - Los strings son **inmutables** (no se pueden modificar directamente).

Operaciones comunes sobre strings:

- `+` → Concatenar strings
- `*` → Repetir un string
- `in`, `not in` → Verificar pertenencia
- `len()` → Obtener largo del string

```python
print("Hola" + "Mundo")     # 'HolaMundo'
print("ja" * 3)             # 'jajaja'
print("a" in "casa")        # True
print(len("universidad"))   # 11
```

Transformaciones comunes:

```python
palabra = "Hola"
print(palabra.upper())  # 'HOLA'
print(palabra.lower())  # 'hola'
```

Comparaciones de strings:

```python
print("gato" != "perro")  # True
print("hola" == "HOLA")  # False
```

5.1. Recorriendo Strings

Los strings pueden recorrerse carácter por carácter usando ciclos `for` o `while`.

Con ciclo for:

```python
palabra = "hola"
for letra in palabra:
    print(letra)
```

Con ciclo while:

```python
palabra = "hola"
i = 0
while i < len(palabra):
    print(palabra[i])
    i += 1
```

Comparación:

- `for` es más simple y claro cuando se recorre una colección.
- `while` requiere un contador manual.

6. Laboratorio 5: Funciones

Funciones en Python

Una **función** es un bloque de código reutilizable que realiza una tarea específica. Permiten estructurar el código, evitar repeticiones y mejorar la organización.

Definición de funciones

Se define una función usando la palabra clave `def`:

```
def saludar():
    print("Hola!")
```

Llamar a una función:

```
saludar()  # Imprime: Hola!
```

Funciones con parámetros

Una función puede recibir datos de entrada (argumentos):

```
def saludar_persona(nombre):
    print("Hola", nombre + "!")
```

Ejemplo de uso:

```
saludar_persona("Python")
# Imprime: Hola Python!
```

Funciones que devuelven valores

Una función puede retornar un valor usando `return`:

```
def cuadrado(numero):
    return numero ** 2
```

Ejemplo de uso:

```
resultado = cuadrado(5)
print(resultado)  # Imprime: 25
```

Diferencia entre print y return

print():
- Muestra información en la pantalla.
- No guarda ni entrega ningún valor usable dentro del programa.

return:

- Devuelve un valor desde la función al programa principal.
- El valor puede ser guardado, modificado o utilizado en operaciones.

Ejemplo correcto usando return:

```
def doble(n):
    return n * 2

resultado = doble(5)
print("El doble es:", resultado)  # Imprime: El doble es: 10
```

Ejemplo incorrecto usando print dentro de la función:

```
def doble(n):
    print(n * 2)

resultado = doble(5)
print("Resultado:", resultado)  # Resultado: None
```

Conclusión: Siempre que necesitemos que una función calcule un resultado y lo entregue al programa, **usamos `return` y no `print`** dentro de la función.

Variables Locales y Globales

Variables locales: Son aquellas que existen solamente dentro de una función. No pueden ser utilizadas fuera de ella.

Variables globales: Son aquellas definidas fuera de cualquier función y pueden ser accedidas desde cualquier parte del programa.

Ejemplo de variable local:

```
def saludar():
    mensaje = "Hola"   # variable local
    return mensaje

print(saludar())
# print(mensaje)  # Error: 'mensaje' no existe fuera de la funcion
```

Ejemplo de variable global:

```
nombre = "Python"  # variable global

def saludar():
    return "Hola " + nombre

print(saludar())  # Imprime: Hola Python
```

Recomendación: Aunque se puede acceder a variables globales dentro de funciones, **se recomienda pasar todo lo necesario como parámetro**, para mantener las funciones independientes y reutilizables.