Volver

UD1 - 1.2

Práctica con un lenguaje

Volver

Rafael Alberti
Instituto de Educación Secundaria

Volver

Indice

Volver

1. Bloques de un programa

Volver

1.1 En Kotlin: piezas básicas

- Un programa se organiza en **paquetes** e **imports**.
- El **punto de entrada** es la función `main`.
- **Funciones**: bloques reutilizables con nombre.
- **Clases** y **objetos** modelan datos y comportamiento.
- Estructuras básicas: secuencia, decisión y repetición.

Volver

1.2 En Python: módulos y ejecución

- Un **programa** es un **módulo** (`.py`) con código ejecutable.
- Puede **definir** funciones, clases y variables.
- `if __name__ == "__main__":` marca código principal.
- **Importaciones** para reutilizar otros módulos.
- Misma lógica: secuencia, decisión y repetición.

Volver

1.2 En Python: módulos y ejecución

```python
# file_one.py
from file_two import function_three

print(f"__name__ es: {__name__}")

def main():
    print("Cuerpo principal")

if __name__ == "__main__":
    main()           # Ejecutado si lanzas este archivo
else:
    print("Importado desde otro módulo")
```

Volver

2. Comenzando con Python

Volver

2.1 Python: visión general

- **Multiparadigma**: imperativo, POO y funcional.
- **Multiplataforma**: Windows, Linux, macOS.
- **Tipado dinámico** y **fuerte**.
- **Interpretado**: ejecuta línea a línea.
- Sintaxis clara y legible.

Volver

2.2 Intérprete interactivo

- Comandos `python` o `python3` abren el REPL.
- Permite probar **expresiones** y **sentencias**.
- `print('Hola')` muestra texto.
- `quit()` para salir.
- Útil para aprender y validar ideas.

```
>>> 2 + 3
5
>>> print('¡Hola mundo!')
¡Hola mundo!
```

Volver

2.3 Primer programa en archivo

- Crea `suma.py` con instrucciones.
- Ejecuta: `$ python3 suma.py`.
- El intérprete lee y ejecuta el código.
- Es la forma más común de trabajar.

Volver

2.3 Primer programa en archivo

```
# suma.py
suma = 2 + 3
print(suma)    # 5
```

Volver

3. Variables, literales y constantes

Volver

3.1 Valores y tipos básicos

- Literales: números, textos y booleanos.
- `int`, `float`, `str`, `bool`, `None`.
- `type(x)` revela el tipo en tiempo de ejecución.
- Ojo: `'17'` es `str`, no `int`.
- Los decimales usan **punto** (`3.2`).

```
>>> type('Hola')   # str
>>> type(17)       # int
>>> type(3.2)      # float
```

Volver

3.2 Variables y asignación

- Una **variable** nombra y referencia un valor.
- Asignación con = crea o actualiza valores.
- El tipo **va con el valor**, no con el nombre.
- `print(x)` muestra su contenido.

```
mensaje = "Texto"
n = 17
pi = 3.14159
print(n, type(n))      # 17 <class 'int'>
```

Volver

3.3 Constantes en Python

- Python **no** tiene constantes de lenguaje.
- Se usa CONVENCIÓN: nombres en MAYÚSCULAS.
- Constantes integradas: `True`, `False`, `None`.
- La inmutabilidad real requiere tipos inmutables.

Volver

4. Operadores, expresiones y sentencias

Volver

4.1 Operadores y precedencia (PEMDSR)

- Paréntesis → Exponenciación → Mult/Div → Suma/Resta.
- Operadores se evalúan de **izq. a der.** con igual nivel.
- Usa paréntesis si hay dudas de orden.

```
2 * (3 - 1)    # 4
(1 + 1) ** 3   # 8
6 + 4 / 2      # 8.0
```

Volver

4.2 Cadenas y operadores

- `+` concatena cadenas: `"a" + "b"`.
- `*` repite cadenas: `"ha" * 3`.
- Mezclar tipos sin convertir lanza error.

```
print('Hola ' + 'Python')    # Hola Python
print('Na' * 4)              # NaNaNaNa
```

Volver

4.3 Expresiones y sentencias

- **Expresión**: combina operandos y operadores y devuelve valor.
- **Sentencia**: instrucción que realiza una acción.
- Ej.: asignación, `if`, `for`, `while`, `def`.

```
a = 2 + 3                # sentencia con expresión
b = a < 10               # expresión booleana
```

Volver

5. Entrada de usuario y comentarios

Volver

5.1 `input()` y conversión

- `input()` lee texto del usuario.
- Siempre devuelve una **cadena**.
- Convierte con `int()`, `float()`.
- Maneja errores si la conversión falla.

```
nombre = input('¿Nombre? ')
edad = int(input('¿Edad? '))
print(f"Hola, {nombre}. Tienes {edad}.")
```

Volver

5.2 Comentarios y docstrings

- Comentario de línea con #.
- Multilínea: varias líneas con #.
- **Docstrings** (""" . . . """) documentan funciones.
- IDEs usan docstrings para ayuda contextual.

```
def suma(a, b):
    """Devuelve la suma de a y b."""
    return a + b
```

Volver

6. Palabras reservadas y nombres

Volver

6.1 Palabras reservadas

- No pueden ser nombres de variables o funciones.
- Ej.: `and`, `class`, `def`, `for`, `if`, `None`, `True`.
- Evita sombras con nombres parecidos.

Volver

6.2 Convenciones de nombres

- Identificadores: letras, dígitos, _; no empiezan con dígito.
- Variables y funciones: `snake_case`.
- Clases: `CamelCase`.
- Python distingue mayúsculas/minúsculas.

Volver

7. Depuración y errores comunes

Volver

7.1 Sintaxis y tiempo de ejecución

- `SyntaxError`: texto inválido (p. ej., espacios en nombre).
- `NameError`: usar nombre no definido.

- Sensible a mayúsculas: `latex` ≠ `LaTeX`.
- Errores semánticos: resultado incorrecto sin fallo.

```
# NameError por variable mal escrita
principal = 100
interest = principle * 0.05
```

Volver

7.2 Indentación y bloques

- Python usa **indentación** para agrupar código.
- Bloque empieza al aumentar sangrado.
- Termina al volver al nivel anterior.
- Usa espacios (4) mejor que tabuladores.

```
def suma(nums):
    total = 0
    for n in nums:
        total += n
    return total
```

Volver

8. Operadores en Python (resumen)

Volver

8.1 Lógicos y comparación

- Lógicos: `and`, `or`, `not`.
- Comparación: `>`, `>=`, `<`, `<=`, `==`, `!=`.
- Encadenables: `1 < x < 20`.
- `and` / `or` devuelven **operandos**, no `True/False`.

```
x, y = 0, 10
x or y   # 10
x and y  # 0
```

Volver

8.2 Aritméticos y asignación

- `+ - * / // % **` (división `/` siempre `float`).
- Asignación compuesta: `+=`, `-=`, `*=`, `/=`, `//=`, `%=`.
- Usa paréntesis para priorizar.

```
x = 7; y = 2
x / y    # 3.5
x // y   # 3
x %= y   # x = 1
```

Volver

8.3 Pertenencia, identidad y bits

- Pertenencia: `in`, `not in`.
- Identidad: `is`, `is not` (mismo **objeto**).
- Bit a bit: `|  ^  &  <<  >>  ~` (entornos numéricos).

```
lista = [1, 3, 2]
3 in lista      # True
x is y          # Mismo objeto en memoria
```

Volver

9. Cierre y práctica guiada

Resumen y mini-retos

- Flujo: intérprete, archivo, ejecución.
- Variables, tipos y conversión.
- Operadores, expresiones y sentencias.
- Entrada con `input()` y docstrings.
- Estilo: nombres y sangrado.

Retos rápidos:

1. Lee nombre y edad, imprime en una línea.
2. Pide 2 números y muestra suma y media.
3. Dado un entero, imprime “par”/“impar”.

Volver

Dudas

Volver

¡Gracias por su atención!

Volver