

upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ARQUITECTURA DE REDES, SISTEMAS Y SERVICIOS
Área de Ingeniería Telemática

Programación procedimental en Java

Area de Ingeniería Telemática
<http://www.tlm.unavarra.es>

Arquitectura de Redes, Sistemas y Servicios

Java



- Orientado a objeto
- Python también
- Dejamos esa parte del lenguaje para “Laboratorio de Programación”
- Vamos a ver simplemente una trasposición de lo que sabéis hacer en Python
- Objetivos:
 - No olvidarse de la programación
 - Introducción a lo que se verá en LP
 - [Teleco] Introducción para programación en red en “Redes de Ordenadores” (sockets)

Sintaxis: Sentencias y bloques

- En Java las sentencias terminan con punto y coma ;
- En Python delimitamos un bloque mediante tabulación, en Java es recomendada, pero no tiene significado
- En Java los bloques vienen delimitados por llaves { }
- Ejemplo:

```
int x = 23;
for (int num = 1; num < 100; num++) {
    if ( (num % 2 == 0) || (num %23 == 0) )
        System.out.println(num + "es par o multimplo de 23");
}
```

Introducción

- El algoritmo codificado en Python podría ser

```
1 x = int(input())
2 minimo = x
3 for i in range(6):
4     if x < minimo:
5         minimo = x
6     x = int(input())
7 print('El número mínimo es', minimo)
```

VARIABLES Y TIPOS DE DATOS

- En Python las variables se adaptan al tipo de dato introducido en ellas
- En Java hay que “declarar” cada variable antes de usarla, indicando su tipo
- Su alcance es el bloque en el que se declararon y los contenidos en él
- Tipos básicos: int, short, long, double, float, boolean, byte, char
- Ejemplos:

```
boolean cierto = true;
int x;
float num = 5674.26;
```

Type	Size	Range
byte	1 byte	-128 to 127
short	2 bytes	-32,768 to 32,767
int	4 bytes	-2,147,483,648 to 2,147,483,647
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	$-3.40282347 \times 10^{38}$ to 3.4028347×10^{38}
double	8 bytes	$-1.79769313486231570 \times 10^{308}$ to $1.79769313486231570 \times 10^{308}$
char	2 bytes	one character
String	2 or more bytes	one or more characters

Tipos de datos y variables: tipos simples

- Sólo guardan un único valor a la vez
- Tipos:

• **bool** (lógicos)

True

Una variable de tipo **bool** sólo puede almacenar los valores **True** o **False**

• **int** (enteros)

-23

Una variable de tipo **int** almacena números **enteros** (tanto positivos como negativos, además del cero).

• **float** (reales)

56734.28

Una variable de tipo **float** almacena tanto números **enteros** como números **con decimales**.

Otros tipos

- Son "objetos" pero los manejaremos sin preocuparnos todavía mucho de eso
- De hecho, en Python también habéis estado usando clases y objetos
- El más típico que usaremos: `String`

Operadores aritméticos

- Los tradicionales: + - * / %
- No hay // para división entera (se hace si ambos son enteros) ni ** para potencias
- Compuestos: += -= *= /= %=
- (pre/post) Incremento y decremento: ++ --
- Ejemplos:

```
x += 1;
```

```
x++;
```

```
System.out.println(--x);
```

Operadores aritméticos

- Con datos de tipo numérico (*int*, *float*) podemos realizar operaciones matemáticas:

Operación	Símbolo	Utilización	Resultado	Tipo numérico	Módulo math
Suma	+	5 + 2	7	int y float	No
Resta	-	4 - 6	-2	int y float	No
Multiplicación	*	3 * 5	15	int y float	No
División entera	//	9 // 2	4	int	No
División real	/	9 / 2	4.5	int y float	No
Módulo de la división	%	9 % 2	1	int	No
Potencia	** pow	2**3 pow(2,3)	8	int y float	No
Valor absoluto	abs()	abs(-5)	5	int y float	No
Raíz cuadrada	sqrt()	sqrt(9)	3	int y float	Sí
Redondeo	round()	round(5.5)	6	float	No

Relacionales y lógicos

- Relacionales: == != > < >= <=
- Lógicos:

and → &&

or → ||

not → !

Operadores relacionales

- Comparan datos de todos los tipos vistos
- Generalmente se usan con datos de tipo simple (int, float)
- El resultado es siempre de tipo **bool** (True o False)

Operación	Símbolo	Utilización	Resultado
Igual	==	5==5	True
		5==7	False
Distinto	!=	5 !=5	False
		5 !=7	True
Mayor o igual	>=	7>=7	True
		7>=5	True
Menor o igual	<=	7<=7	True
		7<=5	False
Mayor	>	7>7	False
		7>5	True
Menor	<	7<7	False
		7<5	False

Operaciones Entrada/Salida

- La salida por pantalla, que es la más simple:

```
System.out.println("Una cadena");  
System.out.println("Las cadenas " + "son sumables.");  
System.out.println("Convirtiendo " + num + " numeros");  
System.out.printf("Con formato el numero %d\n", numero);
```

- La lectura o escritura de/a fichero la vemos en ejemplos (hay múltiples formas de hacerla)

Instrucciones básicas. Operaciones E/S

- **Salida → función print()**
 - Esta función muestra en pantalla lo que se le pase como argumento. Puede mostrar texto literal, valores contenidos en variables y una mezcla de texto y valores contenidos en variables
 - Puede incluir modificadores como 'sep' o 'end'

Ejemplos:



The screenshot displays three examples of Python code and their output. Each example consists of a code editor window and a 'Run' console window.

- Example 1:** Code: `print('Esta línea se imprimirá tal cual')`, `x = 7`, `print(x)`, `print('El número', x, 'está contenido en la variable x')`. Output: `Esta línea se imprimirá tal cual`, `7`, `El número 7 está contenido en la variable x`.
- Example 2:** Code: `print('Hola', '2', 'hasta pronto', sep='*.*.*')`, `print('La', 'casa', 'de mi tía', 'es muy', 'grande', sep='.')`. Output: `Hola*.*.*2*.*.*hasta pronto`, `La.casa.de mi tía.es muy.grande`.
- Example 3:** Code: `print('La casa es blanca')`, `print('con ventanas azules')`, `print('El camino sigue y sigue ', end='')`, `print('desde la puerta')`. Output: `La casa es blanca`, `con ventanas azules`, `El camino sigue y sigue desde la puerta`.

Sintaxis if-else

- Condición entre paréntesis

```
if (condicion) {  
    // A ejecutar si la condición es verdadera  
} else if (otra_condicion) {  
    // A ejecutar si la otra condición es verdadera  
} else {  
    // Ejecutar si ninguna de las anteriores es verdadera  
}
```

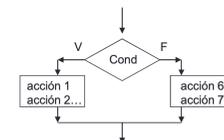
- Ejemplo:

```
if (x > 0) {  
    System.out.println("Positivo");  
} else if (x == 0) {  
    System.out.println("Cero");  
} else {  
    System.out.println("Negativo");  
}
```

Estructuras de control alternativas

- Sentencia if-else

- Permite ejecutar una serie de instrucciones si la condición es verdadera y otro conjunto de instrucciones si la condición es falsa



```
if condición:  
    acciones  
else:  
    otras acciones
```

- Todas las instrucciones que están entre la sentencia **if** y la sentencia **else** deben estar tabuladas a la derecha
- Todas las instrucciones que se tengan que ejecutar solo si la condición es falsa deben estar también tabuladas a la derecha
- Cualquier instrucción que venga después del **else**, que no esté tabulada a la derecha, se ejecutará siempre

Sintaxis switch-case

```
switch (expresion) {  
    case valor1:  
        // Código a ejecutar si expresion == valor1  
        break; // Salir del switch  
    case valor2:  
        // Código a ejecutar si expresion == valor2  
        break; // Salir del switch  
    default:  
        // A ejecutar si no coincide con ningún case  
}  
}
```

Sintaxis: while

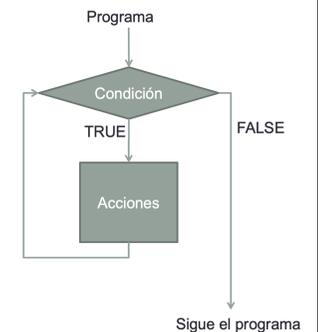
```
while (condicion) {  
    // Instrucciones  
}
```

- **Tenemos** `break` y `continue`

Iterativas: while

- Esquema WHILE:
 - Una o varias instrucciones que se ejecutan mientras se cumpla una condición

```
while condicion:  
    instrucción_1  
    instrucción_2  
    ...
```



Sintaxis: for

- Tiene una versión parecida a Python (una especie de “for each”) pero el tradicional es distinto:

```
for (inicialización; condición; incremento) {  
    // A ejecutar mientras la condición sea verdadera  
}
```

- Ejemplo:

```
int x;  
for (x = 1; x < 10; x++) {  
    System.out.println(x);  
}
```

Iterativas: for-in

- La sentencia **for** de Python itera sobre los elementos de cualquier secuencia (cadenas, listas, conjuntos...)
- A diferencia de **while**, que se puede utilizar siempre, **for** sólo se puede utilizar si conocemos el número de iteraciones a realizar
- Su esquema es:

```
for variable in secuencia:  
    instrucción_1  
    instrucción_2  
    ...
```

Secuencia: colección de valores (generalmente más de uno). Determina el número de veces que se va a repetir el bucle **for**

Variable: variable auxiliar del bucle que irá tomando cada uno de los valores de la secuencia

- Ejemplo:

```
1 for x in ['Pepe', 'Ana', 47]:  
2     print('Hola ', x)  
3 for x in 'Esto es una cadena':  
4     print(x, end='')  
5     print()  
6 for x in ('a', 'e', 'i', 'o', 'u', 38):  
7     print(x)
```

```
Run: forEjemplo <x>  
Hola Pepe  
Hola Ana  
Hola 47  
Esto es una cadena  
u  
e  
38  
a
```

Arrays

- Son objetos
- Las listas en Python son lo más parecido
- Hay que declarar la variable y “construirla”:

```
tipo[] mivariable;  
mivariable = new tipo[cantidad];
```
- Ejemplo:

```
int[] misCienEnteros;  
misCienEnteros = new int[100];
```
- Contenido mutable, tamaño fijo, todos de igual tipo
- Acceso: `mivariable[posicion]`
- Posición de 0 a elementos – 1
- Multidimensionales:

```
double[][] nums = new double[10][20];
```
- Para tamaño variable: ArrayList

Listas

- List
 - Una lista en Python es una secuencia mutable y ordenada de elementos de cualquier tipo, incluso otras listas.
 - Las listas se definen mediante corchetes y los elementos van separados por comas
 - `a = [10, 'hola', 3.7, 1]`

• Permite el acceso directo a cualquier elemento a través de su posición

- Cambiar un elemento `a[2]=45`

• Consultar un elemento `print(a[3]) → 1`

Estructura básica

```
public class MiClasePrincipal {  
  
    public static void main(String[] args) {  
  
        // Código de la función principal  
  
    }  
}
```

Funciones

- Las declararemos en la clase principal
- Ejemplo:

```
public class MiClasePrincipal {  
  
    public static double miFunc(double a, int k) {  
        // Código de la función y retorno:  
        return a*k;  
    }  
  
    public static void main(String[] args) {  
        // Código de la función principal  
        // Usar la función:  
        MiFunc(3.2, 17);  
    }  
}
```