

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
- Vamos a ver lo necesario para adaptar de Python a Java
- Java el detalle se verá en “Laboratorio de Programación” (LP)
- Objetivos:
 - No olvidarse de la programación
 - Introducción a lo que se verá en LP
 - Utilización en las prácticas
 - 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 multiplo de 23");  
}
```

- 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 $3.40282347 \times 10^{38}$
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.

Python (1º curso)

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`
- Otros frecuentes:
 - `Arrays`
 - `ArrayList` y `HashMap`

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

Python (1º curso)

Relacionales y lógicos

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

and → & &

or → | |

not → !

Operadores relacionales

Python (1º curso)

- 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:

```
EjemplosPrint.py  
1 print('Esta línea se imprimirá tal cual')  
2 x = 7  
3 print(x)  
4 print('El número',x,'está contenido en la variable x')
```

```
Run: EjemplosPrint  
Esta línea se imprimirá tal cual  
7  
El número 7 está contenido en la variable x
```

```
1 print('Hola', '2', 'hasta pronto',sep='...')  
2 print('La','casa','de mi tía','es muy', 'grande',sep='.')
```

```
Run: EjemplosPrint  
Hola...2...hasta pronto  
La.casa.de mi tía.es muy.grande
```

```
EjemplosPrint.py  
1 print('La casa es blanca')  
2 print('con ventanas azules')  
3 print('El camino sigue y sigue ',end='')  
4 print('desde la puerta')
```

```
Run: EjemplosPrint  
La casa es blanca  
con ventanas azules  
El camino sigue y sigue desde la puerta
```

Python (1º curso)

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  
}
```

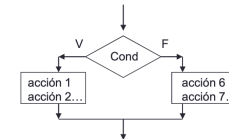
- Ejemplo:

```
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

Python (1º curso)

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  
}  
}
```

- Versiones modernas de Python tienen un equivalente (`match-case`)

Sintaxis: while

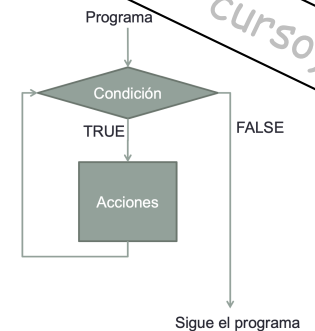
```
while (condicion) {  
    // Instrucciones  
}  
do {  
    // Instrucciones  
} while (condicion);
```

- Tenemos break y continue

Iterativas: while

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

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



Python (1º curso)

Sintaxis: for

- Java tiene el `for` "tradicional" (C-style) que es diferente a Python:

```
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
    ...
```

- 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)
```

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

Run: forEjemplo.py

```
Hola Pepe
Hola Ana
Hola 47
Esto es una cadena
a
e
i
38
```

Python (12 curso)

Sintaxis: for

- También tiene `for-each` para “colecciones”

```
for (tipo variable: coleccion) {
    // A ejecutar para cada elemento
}
```

- Ejemplo:

```
int[] values = {1, 10, 100};
for (int v: values) {
    System.out.println(v);
}
```

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
    ...
```

- 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)
```

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

Run: forEjemplo.py

```
Hola Pepe
Hola Ana
Hola 47
Esto es una cadena
u
o
e
38
a
i
```

Python (12 curso)

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

[0]	tipo
[1]	tipo
[2]	tipo
[3]	tipo
[4]	tipo
[5]	tipo
[6]	tipo
[7]	tipo

Listas

Python (1º curso)

- 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 del programa

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

- En un fichero `MiClasePrincipal.java`
- Compilar con: `javac MiClasePrincipal.java`
- Ejecutar con: `java MiClasePrincipal`
- La ejecución comienza por la función con nombre `main`

Funciones

- Las declararemos en ese fichero (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);  
    }  
}
```

```
def sumaN(v, n):  
    for i in range(len(v)):  
        v[i] += n  
    return v
```

Python (1º curso)

Excepciones

- Para errores y otras situaciones inesperadas
- En lugar de devolver un código de error se “lanza” una
- Encerramos en un bloque `try-catch` el código que puede lanzar una excepción y eso nos permite procesarla
- Si no se “atrapan” terminan el programa
- Ejemplo:

```
int x = 10;
try {
    int r = x / 0;
}
catch (Exception e) {
    System.err.println("Error al dividir");
}
```

- También existen en Python

Clases y objetos

- Se verá en detalle en LP
- Ejemplo: listas y cadenas en Python
- Clase: es una “plantilla” para un nuevo tipo de datos

```
class EthernetFrame {  
    String src;  
    String dst;  
  
    boolean isBroadcast() {  
        return this.dst.equals("ff:ff:ff:ff:ff:ff");  
    }  
}
```

- Objeto:
 - Es una variable cuyo tipo es la clase
 - Es una “instancia” creada a partir de la plantilla

```
EthernetFrame miObjeto = new EthernetFrame();
```

Clases y objetos

- Pueden tener atributos (estado)
 - Son variables contenidas en el objeto
 - Aunque también podrían ser de la clase
- Pueden tener métodos (funcionalidad)
 - Son funciones que actúan sobre el objeto (o la clase)

```
class EthernetFrame {  
    String src;  
    String dst;
```

```
    boolean isBroadcast() {  
        return this.dst.equals("ff:ff:ff:ff:ff:ff");  
    }  
}
```

```
EthernetFrame miObjeto = new EthernetFrame();
```

```
class EthernetFrame:  
    src = ""  
    dst = ""  
  
    def is_broadcast(self):  
        return self.dst == "ff:ff:ff:ff:ff:ff"  
  
miObjeto = EthernetFrame()
```

Python

Objetos

- Se crean instancias de clases con `new`
- Acceso a atributos y métodos con `.`

```
EthernetFrame f1 = new EthernetFrame()  
f1.src = "01:02:03:04:05:06";  
f1.dst = "ff:ff:ff:ff:ff:ff";  
System.out.println(f1.isBroadcast()); // true
```

- Se pueden crear múltiples instancias (objetos) independientes

```
EthernetFrame f2 = new EthernetFrame()  
EthernetFrame f2 = new EthernetFrame()
```

Objetos

- Pueden definir un constructor (o varios)
- Se ejecuta al hacer `new`

```
class EthernetFrame {  
    String src;  
    String dst;  
  
    EthernetFrame(String src, String dst) {  
        this.src = src;  
        this.dst = dst;  
    }  
}  
  
String src = "01:02:03:04:05:06";  
String dst = "ff:ff:ff:ff:ff:ff";  
EthernetFrame f1 = new EthernetFrame(src, dst);
```

Objetos: Ejemplo

```
class Switch {  
    void sendFrame (String iface, EthernetFrame frame) {  
        System.out.printf ("Sending frame from %s: %s -> %s\n",  
            iface, frame.src, frame.dst)  
    }  
}
```

```
EthernetFrame f1 = new EthernetFrame("...", "...");  
EthernetFrame f2 = new EthernetFrame("...", "...");
```

```
Switch sw = new Switch ()
```

```
sw.sendFrame("eth0", f1) ;  
sw.sendFrame("eth1", f1) ;  
sw.sendFrame("eth0", f2) ;
```

Objetos

- La programación orientada a objetos ayuda a estructurar programas grandes con:
 - Modificadores
 - Herencia y polimorfismo
 - Interfaces
 - Clases abstractas
- Hay más detalle de algunos apartados en el manual de introducción a Java
- El resto se verá en “Laboratorio de Programación”