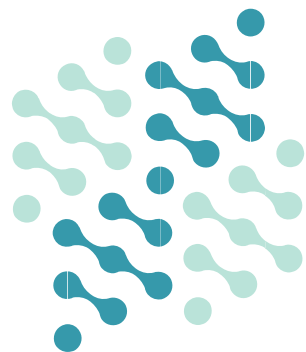


# Introducción a la Computación

Primer Cuatrimestre de 2019



**DEPARTAMENTO  
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA

# Programa

Un **programa** es una secuencia finita de **instrucciones**.

## Ejemplo:

**Ingredientes:** 15 huevos, 600 gramos de harina, 600 gramos de azucar

- 1.- Mientras no estén espumosos, batir los huevos junto con el azúcar,
- 2.- agregar la harina en forma envolvente sin batir,
- 3.- batir suavemente,
- 4.- colocar en el horno a 180 grados,
- 5.- si le clavo un cuchillo y sale húmedo, entonces ir a 4.-
- 6.- retirar del horno,
- 7.- mientras no esté frío, esperar
- 8.- desmoldar y servir

# Instrucción

Una **instrucción** es una operación que:

- transforma los datos (el *estado*), o bien
- modifica el flujo de ejecución.

- 1.- Mientras no estén espumosos, batir los huevos junto con el azúcar,
- 2.- agregar la harina en forma envolvente sin batir,
- 3.- batir suavemente,
- 4.- colocar en el horno a 180 grados,
- 5.- si le clavo un cuchillo y sale húmedo, entonces ir a 4.-
- 6.- retirar del horno,
- 7.- mientras no esté frío, esperar
- 8.- desmoldar y servir

# Variable

Una **variable** es un **nombre** que denota la **dirección** de una celda en la memoria, en la cual se almacena un **valor**.

En esa celda de memoria es posible:

- **leer** el valor almacenado, y
- **escribir** un valor nuevo, que reemplace al anterior.

En C++, cada variable tiene asociado un **tipo** (bool, int, float, char, etc.), por lo cual es necesario **declararlas** antes de usarlas.

Ejemplo en C++:

```
int x;           // Declaro la variable x de tipo int.
x = 10;          // Asigno el valor 10 en la variable x.
cout << x;       // Imprimo en pantalla el valor de x.
```

# Estado

Se denomina **estado** al valor de todas las variables de un programa en un punto de su ejecución.

**Es una “foto” de la memoria  
en un momento determinado.**

# Asignación

***VARIABLE = EXPRESIÓN ;***

Almacena el valor de la *EXPRESIÓN* en la dirección en memoria denotada por *VARIABLE*.

**Ejemplos:**

```
int a;                // Declaración de a.  
array<int, 2> b;      // Declaración de b.  
a = 0;  
b[a] = 1;  
b[b[0]] = 999 * b[a] + a;  
b[b[1]/333-2] = 123;
```

# Ejemplo de repaso

```
int a;
```

```
array<int, 2> b;
```

```
{ a = ↑ ∧ |b| = 2 ∧ b[0] = ↑ ∧ b[1] = ↑ }
```

```
a = 0;
```

```
{ a = 0 ∧ |b| = 2 ∧ b[0] = ↑ ∧ b[1] = ↑ }
```

```
b[a] = 1;
```

```
{ a = 0 ∧ |b| = 2 ∧ b[0] = 1 ∧ b[1] = ↑ }
```

```
b[b[0]] = 999 * b[a] + a;
```

```
{ a = 0 ∧ |b| = 2 ∧ b[0] = 1 ∧ b[1] = 999 }
```

```
b[b[1]/333-2] = 123;
```

```
{ a = 0 ∧ |b| = 2 ∧ b[0] = 1 ∧ b[1] = 123 }
```

# Condicional

**if (CONDICIÓN) { PROG1 }**

*CONDICIÓN* es una expresión que arroja resultado verdadero o falso;  
*PROG1* es un programa.

*PROG1* se ejecuta **si y sólo si** *CONDICIÓN* arroja valor verdadero.

**Ejemplo:**

```
if (1 > 5) {  
    cout << "1 es mayor que 5." << endl;  
}  
if (1 < 5) {  
    cout << "1 es menor que 5." << endl;  
}
```



# Condicional

**if (CONDICIÓN) { PROG1 } else { PROG2 }**

*CONDICIÓN* es una expresión que arroja V o F.

*PROG1* y *PROG2* son programas.

*PROG1* se ejecuta **sii** *CONDICIÓN* arroja valor verdadero.

*PROG2* se ejecuta **sii** *CONDICIÓN* arroja valor falso.

**Ejemplo:**

```
if (1 > 5) {  
    cout << "1 es mayor que 5." << endl;  
} else {  
    cout << "1 es menor que 5." << endl;  
}
```

# Condicional – Otro ejemplo

¿Qué imprime por pantalla este código?

```
int a = 10;
array<int, 2> b = {100, 1};

if (b[0] / (a * 10) == b[1]) {
    b[0] = b[0] - 1;
    b[1] = b[1] * 5;
} else {
    b[0] = b[0] + 1;
    b[1] = b[1] * 3;
}

cout << a << ", " << b[0] << ", " << b[1];
```

Para mayor claridad,  
indentar cada bloque de código.

# Ciclo

**while (CONDICIÓN) { PROG1 }**

*CONDICIÓN* es una expresión que arroja resultado verdadero o falso;  
*PROG1* es un programa.

*PROG1* se ejecuta una y otra vez, **mientras** *CONDICIÓN* siga arrojando valor verdadero.

**Ejemplo:**

```
int i = 0;
while (i < 3) {
    cout << i;
    i = i + 1;
}
```

# Ciclo – Otro ejemplo

**while (CONDICIÓN) { PROG1 }**

**Ejemplo:**

```
int i = 0;
while (i < 3) {
    if (i % 2 == 0) {
        cout << i << " es par" << endl;
    } else {
        cout << i << " es impar" << endl;
    }
    i = i + 1;
}
```

# Ciclos anidados

¿Qué imprime por pantalla este código?

**Ejemplo:**

```
int fil = 1;
while (fil <= 5) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil + 1;
}
```





# Alcance de las variables

El **alcance** (*scope*) de una variable es el código en el cual una variable puede ser accedida.

Su definición varía según el lenguaje.

En C++ el alcance de una variable va:

- desde su **definición**,
- hasta el final del **bloque actual**,
- incluyendo **bloques anidados**.

```
main() {  
    a = 1;   
    int i = 0;  
    while (i < 10) {  
        int a = 1;  
        if (i > 5) {  
            i = i + a;   
        }  
        cout << i + a;   
        i = i + 1;  
    }  
    cout << a;   
}
```

# Evaluación de expresiones lógicas

```
array<int, 3> a = {2, 4, 8};  
int i = 0;  
while (i < a.size() && a[i] % 2 == 0) {  
    cout << a[i];  
    i = i + 1;  
}
```

## Evaluación *Lazy*:

false && *EXP* → false

true || *EXP* → true

En estos casos, la expresión lógica *EXP* **no se evalúa**.

## Otros ejemplos:

¿Cuánto vale (a != 0 && 1/a > 0.1) si a=0? ¿Si a=1?

¿Cuánto vale (a == 0 || 1/a > 0.1) si a=0? ¿Si a=1?

# Azúcar sintáctica en C++

Sintaxis añadida para facilitarle las cosas al programador.

**NO** recomendamos usarla en esta materia.

Azúcar sintáctica en C++ (no recomendada)	Sintaxis recomendada
<code>a++;</code>	<code>a = a + 1;</code>
<code>a--;</code>	<code>a = a - 1;</code>
<code>if (CONDICIÓN) <i>INSTRUCCIÓN</i>;</code>	<code>if (CONDICIÓN) {     <i>INSTRUCCIÓN</i>; }</code>
<code>for (int i=0; i&lt;10; i++) {     <i>PROGRAMA</i> }</code>	<code>int i = 0; while (i &lt; 10) {     <i>PROGRAMA</i>;     i = i + 1; }</code>



# Programa

Un **programa** es una secuencia finita de **instrucciones**.

- Declaración de variables

*TIPO NOMBRE;*

- Asignación

*VARIABLE = EXPRESIÓN;*

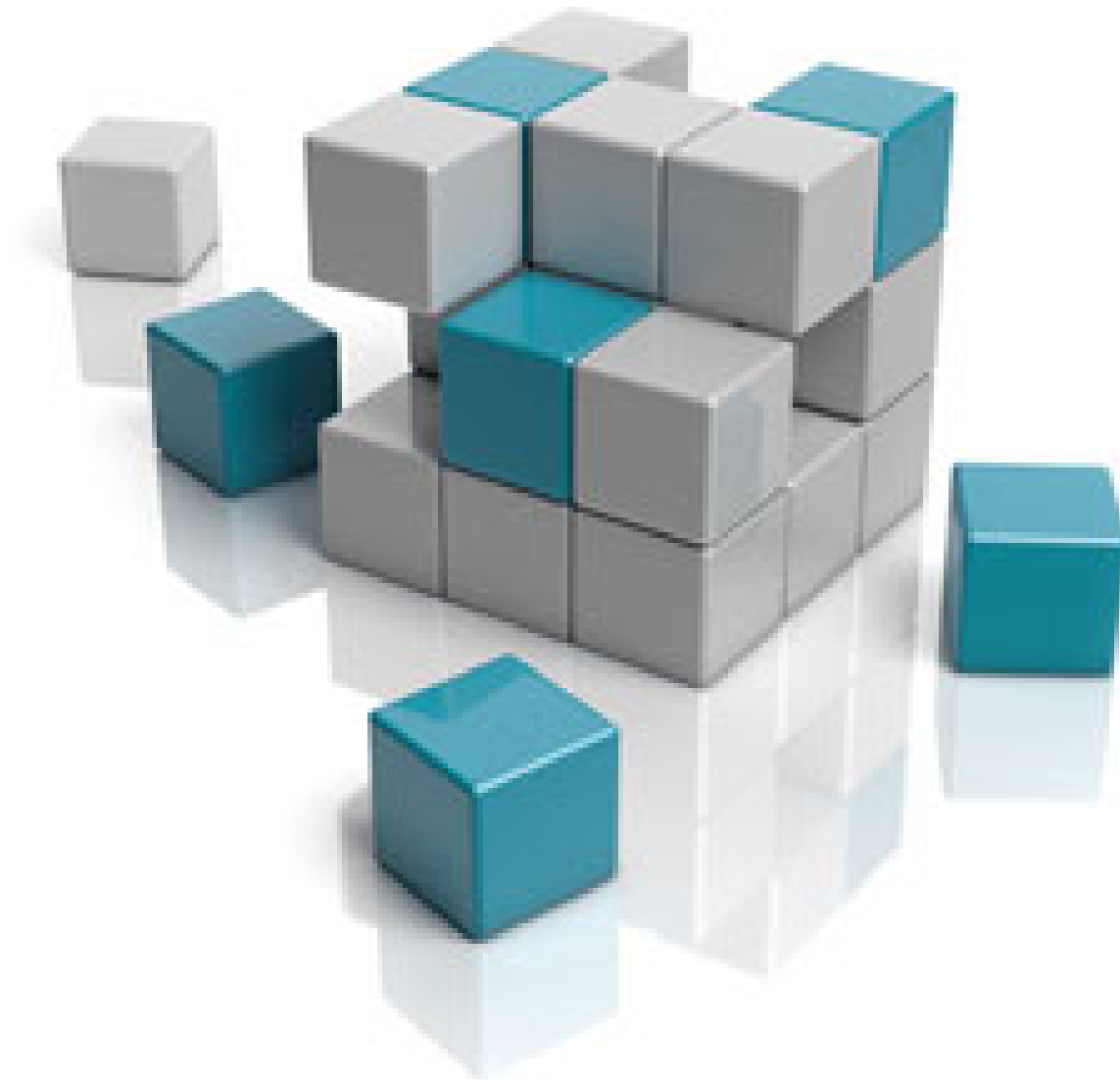
- Condicional

*if (CONDICIÓN) { PROG1 } else { PROG2 }*

- Ciclo

*while (CONDICIÓN) { PROG1 }*

# Modularidad del código: funciones y procedimientos



```

int fil = 1;
while (fil <= 5) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil + 1;
}

```

Salida:

```

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

```

¿Qué podríamos cambiar para lograr esta salida?

```

1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```

?

```

int  fil = 5;
while (fil >= 1) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil - 1;
}

```

Salida:

```

1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```

¿Qué podríamos cambiar para lograr esta salida?

```

1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1
1

```

?

```

int fil = 5;
while (fil >= 1) {
    if (fil % 2 != 0) {
        int col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
        col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
    }
    fil = fil - 1;
}

```

Salida:

```

1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1
1

```

¿Y para esta salida?

```

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1 2 3
1
1
1

```

?

```

int fil = 1;
while (fil <= 5) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil + 1;
}

```

```

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

```

```

int fil = 5;
while (fil >= 1) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil - 1;
}

```

```

1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```

```

int fil = 5;
while (fil >= 1) {
    if (fil % 2 != 0) {
        int col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
        col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
    }
    fil = fil - 1;
}

```

```

1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1
1

```

¿Cómo puedo hacer para reusar este código sin tener que copiarlo una y otra vez?

```
void imprimir_fila(int fil) {  
    int col = 1;  
    while (col <= fil) {  
        cout << col << " ";  
        col = col + 1;  
    }  
    cout << endl;  
}
```

Ejemplos:

`imprimir_fila(2)`      imprime: 1 2

`imprimir_fila(5)`      imprime: 1 2 3 4 5

```

int fil = 1;
while (fil <= 5) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil + 1;
}

```

```

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

```

```

int fil = 5;
while (fil >= 1) {
    int col = 1;
    while (col <= fil) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
    fil = fil - 1;
}

```

```

1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```

```

int fil = 5;
while (fil >= 1) {
    if (fil % 2 != 0) {
        int col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
        col = 1;
        while (col <= fil) {
            cout << col << " ";
            col = col + 1;
        }
        cout << endl;
    }
    fil = fil - 1;
}

```

```

1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1
1

```



```
int fil = 1;
while (fil <= 5) {
    imprimir_fila(fil);
    fil = fil + 1;
}
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
int fil = 5;
while (fil >= 1) {
    if (fil % 2 != 0) {
        imprimir_fila(fil);
        imprimir_fila(fil);
    }
    fil = fil - 1;
}
```

```
1 2 3 4 5
1 2 3 4 5
1 2 3
1 2 3
1
1
```

```
int fil = 5;
while (fil >= 1) {
    imprimir_fila(fil);
    fil = fil - 1;
}
```

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
void imprimir_fila(int n) {
    int col = 1;
    while (col <= n) {
        cout << col << " ";
        col = col + 1;
    }
    cout << endl;
}
```

## Resultado: Código modular.

- Más claro para los humanos.
- Más fácil de actualizar.

(Ej: ¿Qué pasa si ahora quiero separar los números con “,” en lugar de “ ”?)

# Función

Una **función** es una unidad de código que **aísla** una parte de un cómputo. Es un programa dentro de un programa.

- Permite dividir un problema en **problemas más simples**.
- Permite **ordenar** conceptualmente el código para que sea más fácil de entender.
- Permite **reutilizar** soluciones a problemas pequeños en la solución de problemas mayores.

Este es el **tipo** del valor que devuelve la función.

# Función

Estos son los **parámetros** de la función.

```
int raiz_cuadrada(int n) {  
    int i = 1;  
    while (i * i <= n) {  
        i = i + 1;  
    }  
    return (i - 1);  
}
```

Al llegar acá, se evalúa la expresión, se **devuelve** el valor resultante y la función **termina**.

Ahora que tengo definida la función **raiz\_cuadrada**, puedo usarla en otra parte de mi código para construir nuevas expresiones.

Ejemplo:

```
int x = raiz_cuadrada(100);  
x = raiz_cuadrada(x + 6) / 2;
```

# Procedimiento

**Procedimiento** == Función que no devuelve valor alguno.

```
void imprimir_fila(int n) {  
    int col = 1;  
    while (col <= n) {  
        cout << col << " ";  
        col = col + 1;  
    }  
    cout << "\n";  
}
```

En C++, los procedimientos son de tipo **void** (“nulo”, es español).

```
int main() {  
    int x = 1;  
    while (x <= 5) {  
        cout << raiz_cuadrada(x) << "\n";  
        x = x + 1;  
    }  
}  
  
int raiz_cuadrada(int n) {  
    int i = 1;  
    while (i * i <= n) {  
        i = i + 1;  
    }  
    return i - 1;  
}
```

Cada ejecución de una función tiene su **propio espacio de memoria**, como si fuera un programa separado.

**n**, **i** son **alcanzables** dentro de raiz\_cuadrada, pero no fuera.

# Un detalle técnico

En C++, las funciones deben definirse **antes** de ser usadas.

- Antes == Más arriba en el archivo de código.
- Por eso, la función principal (**main**) suele definirse abajo de todo.

Si necesitamos usar una función antes de su definición, podemos copiar su **signatura** arriba de todo:

```
#include <iostream>
using namespace std;
int raiz_cuadrada(int n);
int main() {
    cout << raiz_cuadrada(8) << "\n";
}

int raiz_cuadrada(int n) {
    int i = 1;
    while (i * i <= n) {
        i = i + 1;
    }
    return i - 1;
}
```

# Repaso

- Condicional:

```
if (CONDICIÓN) { PROG }  
if (CONDICIÓN) { PROG1 } else { PROG2 }
```

- Ciclos:

```
while (CONDICIÓN) { PROG }
```

- Modularidad del código: funciones y procedimientos.

```
int raiz_cuadrada(int n) {  
    int i = 1;  
    while (i * i <= n) {  
        i = i + 1;  
    }  
    return (i - 1);  
}
```

tipo de la función

parámetros

valor de retorno

- Alcance de variables (en C++, desde la definición hasta el final del bloque)
- Evaluación lazy de expresiones. Ej: `(a==0 || 1/a>0.1)`
- Próximos temas: Especificación de problemas y verificación de programas.