Parte 1. Fundamentos de la programación

Unai Pérez-Goya
Área de Lenguajes y Sistemas Informáticos

Curso 2024/2025

Parte 1.

Fundamentos de la programación

Sesion III: Estructuras de programación y funciones

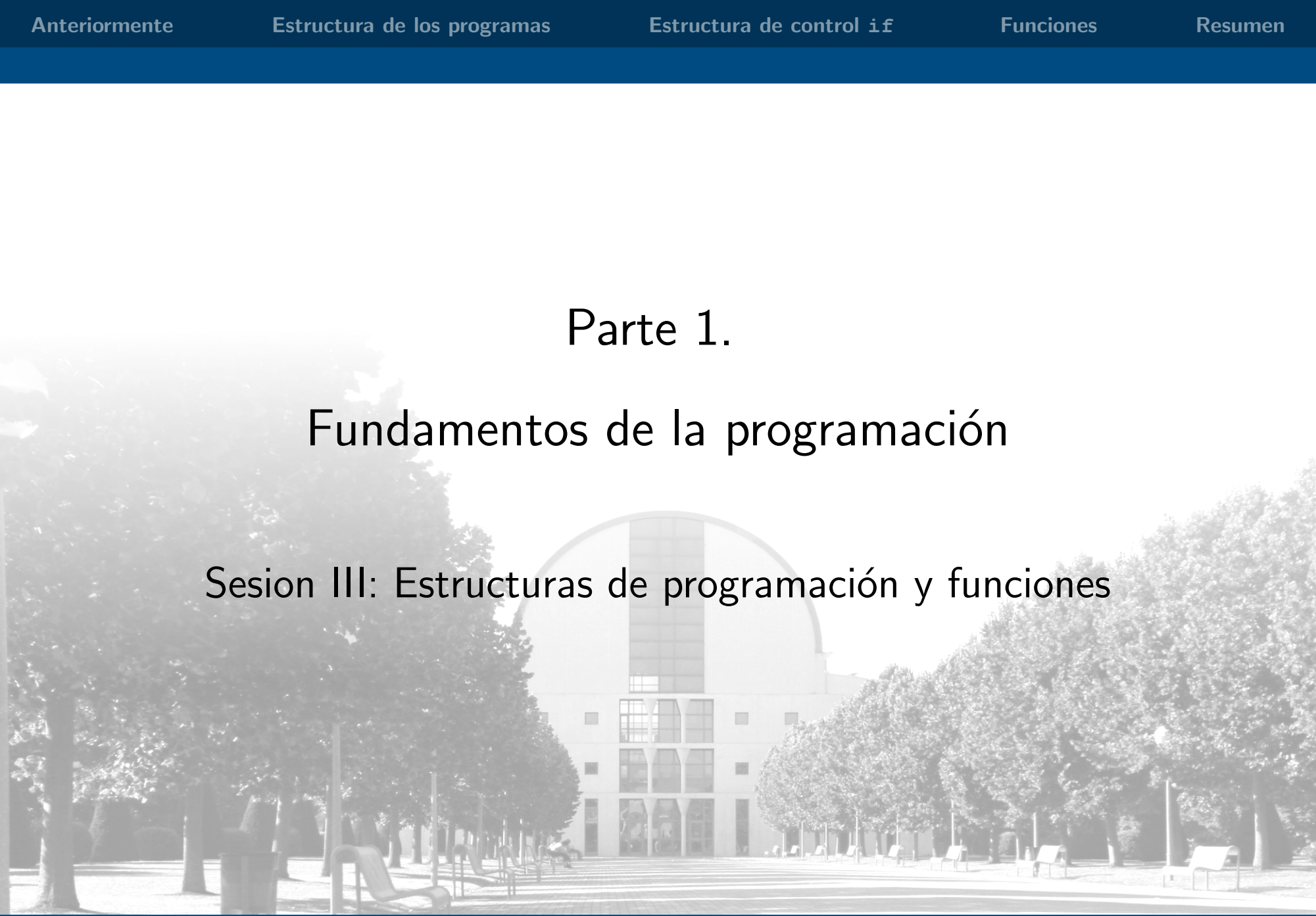


Tabla de contenidos

Resumen de sesiones anteriores

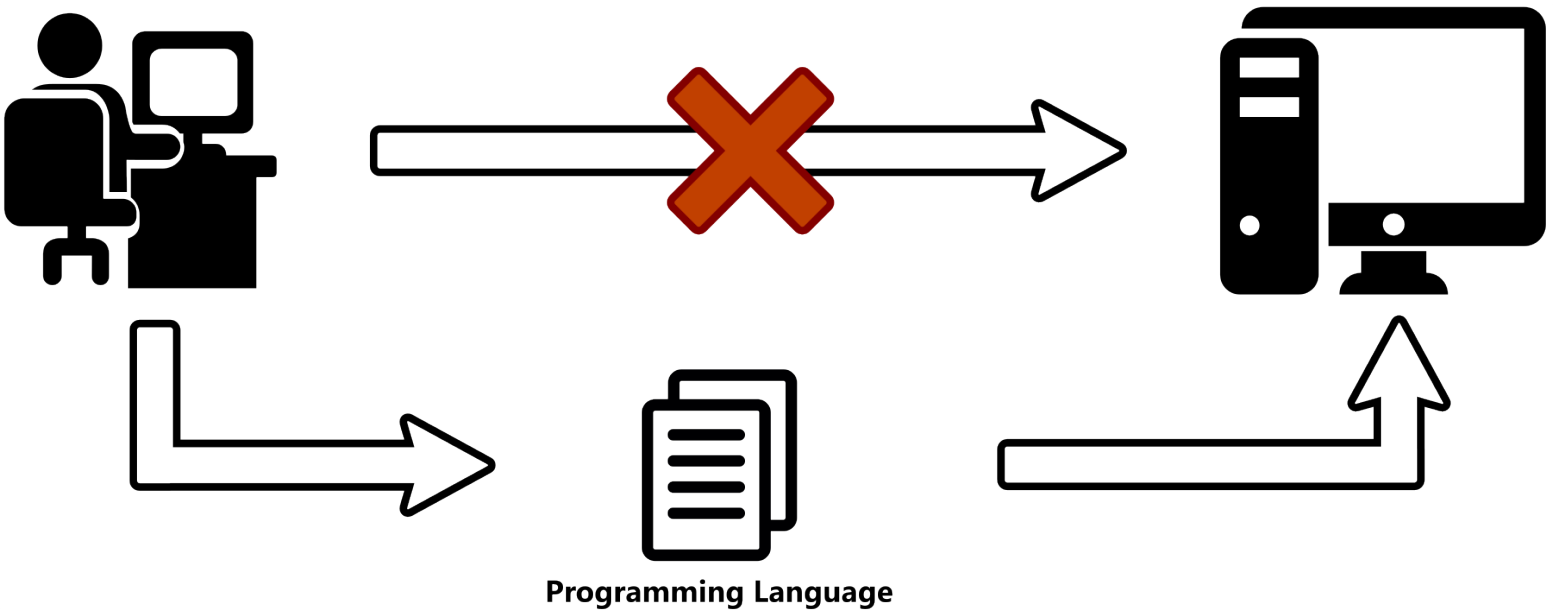
Contenido de la sesión II

Lista de contenidos

1. Introducción a Python. Historia y características;
2. Lenguajes de programación. Concepto y uso;
3. Lenguajes interpretados *vs.* compilados. Programas ejecutables. Máquina intérprete;
4. Entorno de desarrollo integrado (IDE)

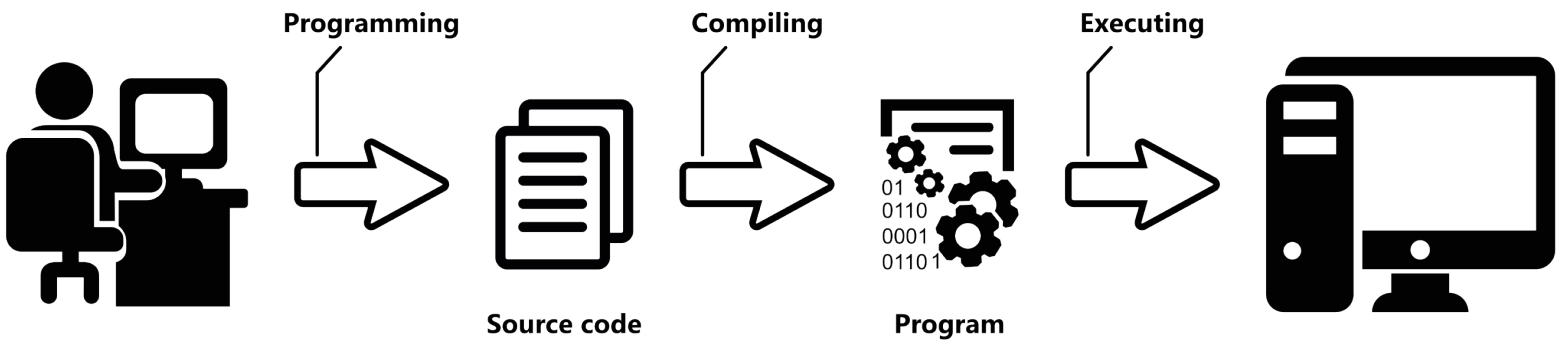
En sesiones anteriores

Lenguajes de programción



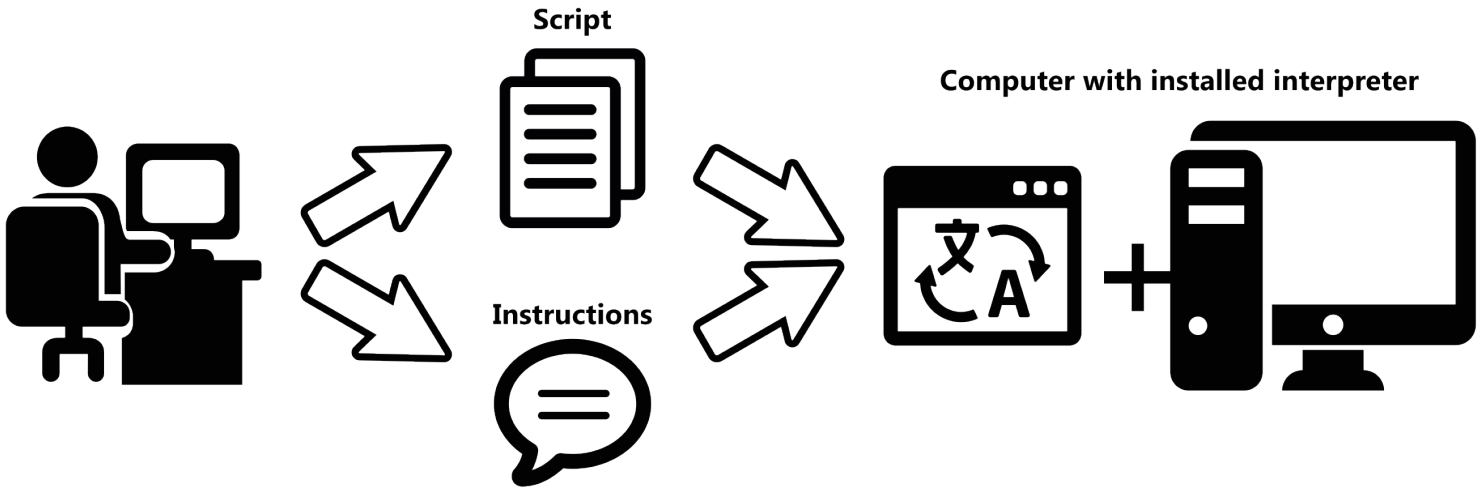
En sesiones anteriores

Proceso de compilación



En sesiones anteriores

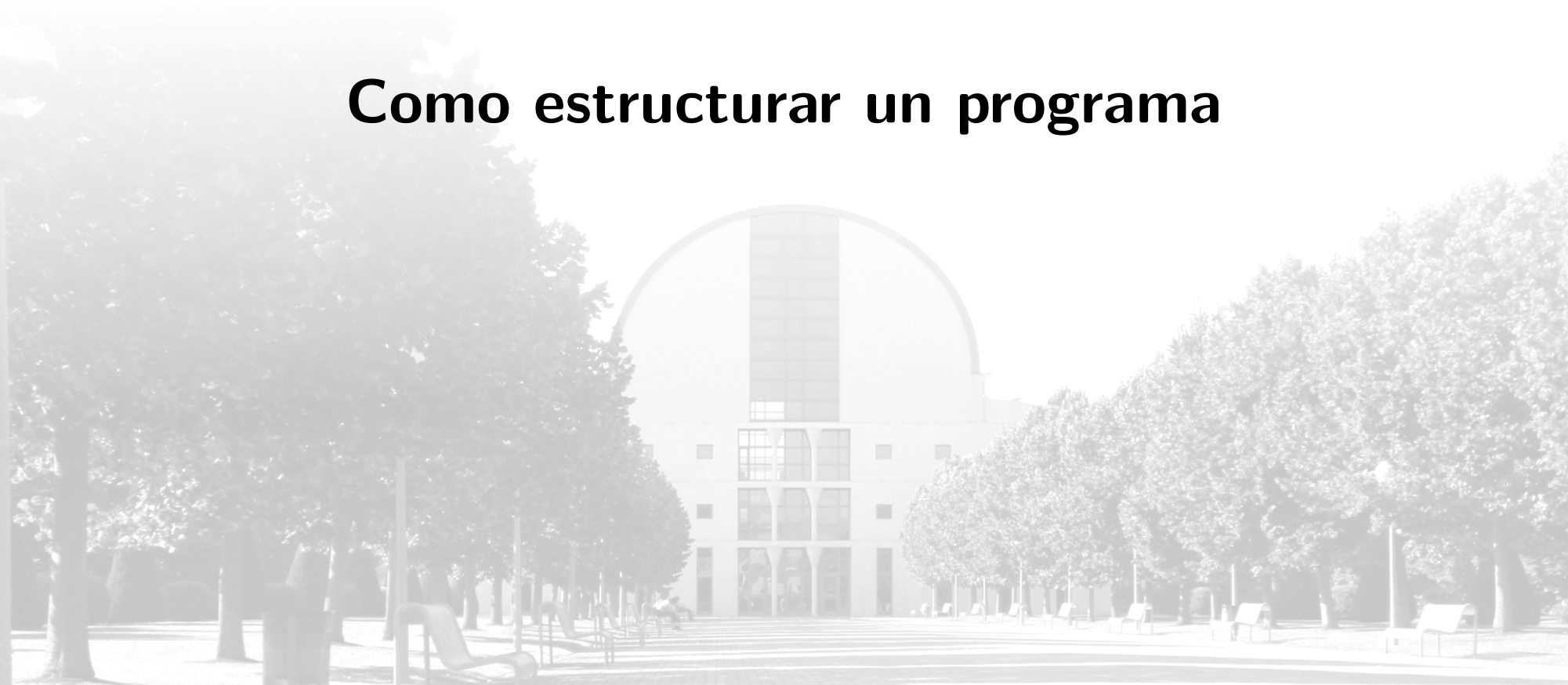
Compilado vs. Interpretado



Contenido de la sesión II

- Variables y uso básico en python
- Declaración y asignación;
- Tipos
  - Numéricos
  - Texto
  - Booleanos
- Constantes

Como estructurar un programa



Orden de ejecución de línea

- Todos los programas, en todos los lenguajes de programación tienen un orden de ejecución.
- Este orden, para python, está definido de izquierda a derecha y de arriba abajo.
- Si la línea contiene la instrucción de asignación, primero se ejecutan el código a la derecha de la asignación
- Por ejemplo,

```
1 # python script
2 mi_valor = 5 + 3
```

Código: Orden de ejecución.

  - Primero se ejecutará 5 + 3
  - Después se guardará el valor 8 en `mi_valor`

Prioridad de ejecución

- Sin embargo, este orden puede ser modificado ya que las operaciones tienen orden de prioridad, igual que en matemáticas
- La multiplicación tiene prioridad sobre la suma

```
1 # python script
2 mi_valor = 1 + 2 * 3
```

Código: Orden de prioridad.

- Los paréntesis tienen prioridad sobre las operaciones

```
1 # python script
2 mi_valor = (1 + 2) * 3
```

Código: Orden de prioridad paréntesis.

Ejecución secuencial

- Además del orden de ejecución de izquierda a derecha, los programas se ejecutan de arriba abajo, es decir, secuencialmente
- La multiplicación tiene prioridad a la suma

```
1 suma = 2 + 2
2 suma_por_2 = suma * 2
```

Código: Orden de ejecución.

- Si no se estructuran y tienen muchas líneas los programas pueden ser caoticos.
- Es necesario seguir una estructuración

Ejecución secuencial

- Crea un programa que resuelva $x^2 - 5x + 6 = 0$.

  - ¿Es correcta la siguiente solución al problema?
  - ¿Es esta la mejor solución?

```
1 print((-5+(5**2-(4*1*6))**0.5)/2*1)
```

Código: Orden de ejecución.

Ejecución secuencial

- Los programas deben diseñarse para resolver el mayor número de casos posibles.
- El programa anterior desde el punto de vista informático no tenía ningún sentido.
- Veamos otra solución:

```
1 a = 1
2 b = 5
3 c = 6
4 print((-b+(b**2-(4*a*c))**0.5)/2*a)
```

Código: Orden de ejecución.

- Fijaos en la estructuración de la aplicación, primero se declaran las variables y luego se resuelve el sistema.

Ejecución secuencial

- Sin embargo todavía se puede organizar mejor...

```
1 a,b,c = 1,5,6
2 x = (-b+(b**2-(4*a*c))**0.5)/2*a
3 print(x)
```

Código: Orden de ejecución.

- En este caso, se han construido 3 bloques en forma de 3 líneas:
  - Declaración de variables
  - Logica de la aplicación o cálculo
  - Mostrar resultado
- Esta estructuración es mucho más lógica y es practicamente la que utilizaremos en esta asignatura.

Estructuración de código: Ejemplo

- Los cuatro bloques en los que se tendrán que organizar todos los programas de la asignatura.

```
1  #----------------Bloque 1 descripcion de la aplicacion
2  # python script
3  # Autor: Unai Perez Goya
4  # Descripcion: Programa para resolver ecuaciones de
       segundo grado
5  #----------------Bloque 2 declaracion de variables
6  a,b,c = 1,5,6
7  #----------------Bloque 3 logica de la aplicacion
8  x = (-b+(b**2-(4*a*c))**0.5)/2*a
9  #----------------Bloque 4 mostrar resultados
10 print(x)
```

Código: Ejemplo de estructuración de código.

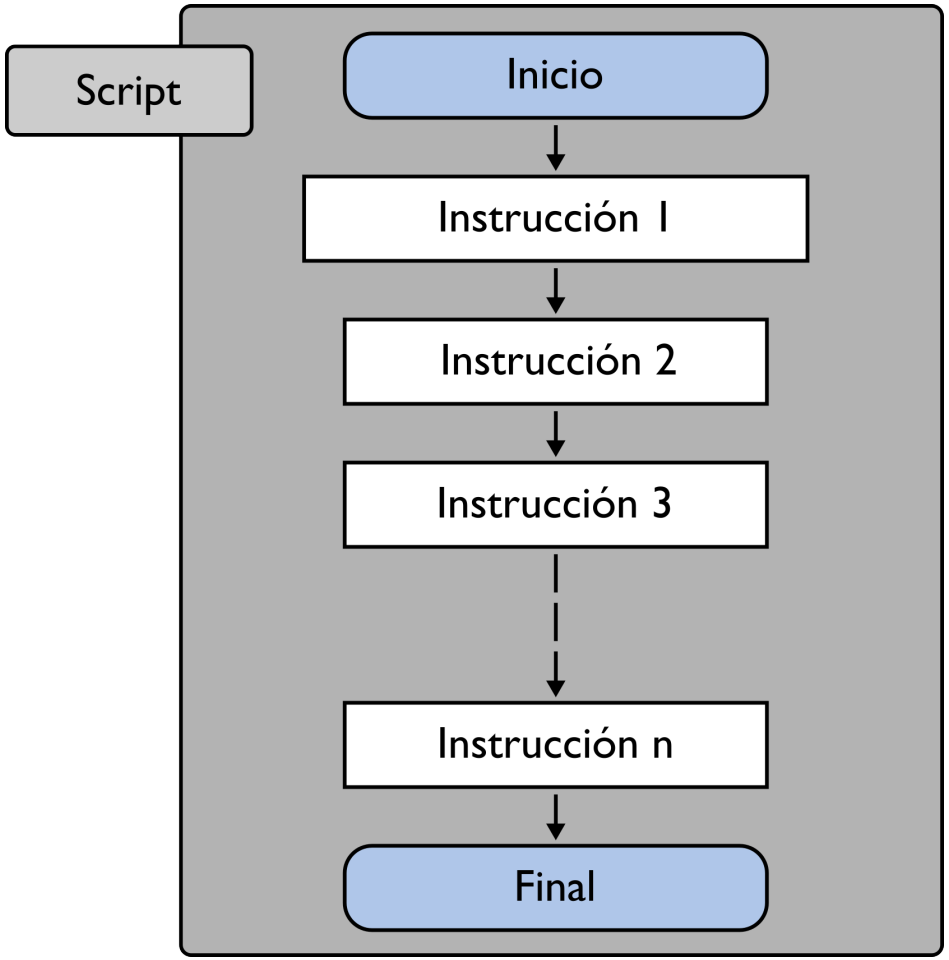
Estructura de control `if`

Programas secuenciales

- Los *programas* utilizados hasta ahora han sido secuenciales, es decir, compuestos por listas de instrucciones;
  - Se ejecutan todas las instrucciones;
  - Ninguna se queda sin ejercutarse.
- Si la instrucción ejecutada es una función, se ejecutará un conjunto de instrucciones.
- Son programas de poca potencia.

Programas secuenciales

Representación gráfica



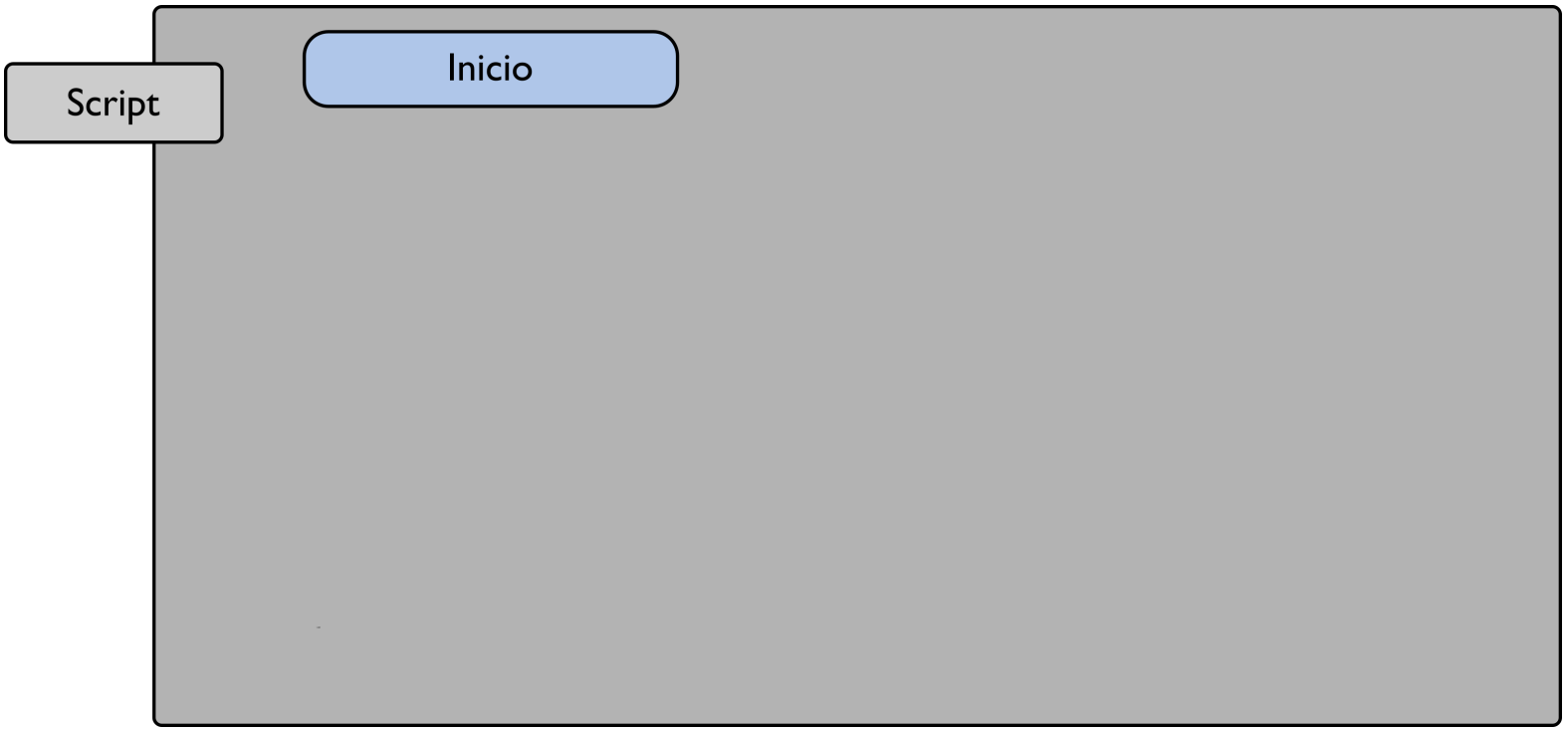
Programas secuenciales

- Los programas secuenciales no pueden:
  - Adaptar a los datos;
  - Adaptar a los usuarios;
  - Es decir, no pueden adaptarse.
- Existen multiples modos para adaptar la ejecución a las diferentes situaciones, puede depender de diferentes factores.
  - La definición técnica es control de flujo.

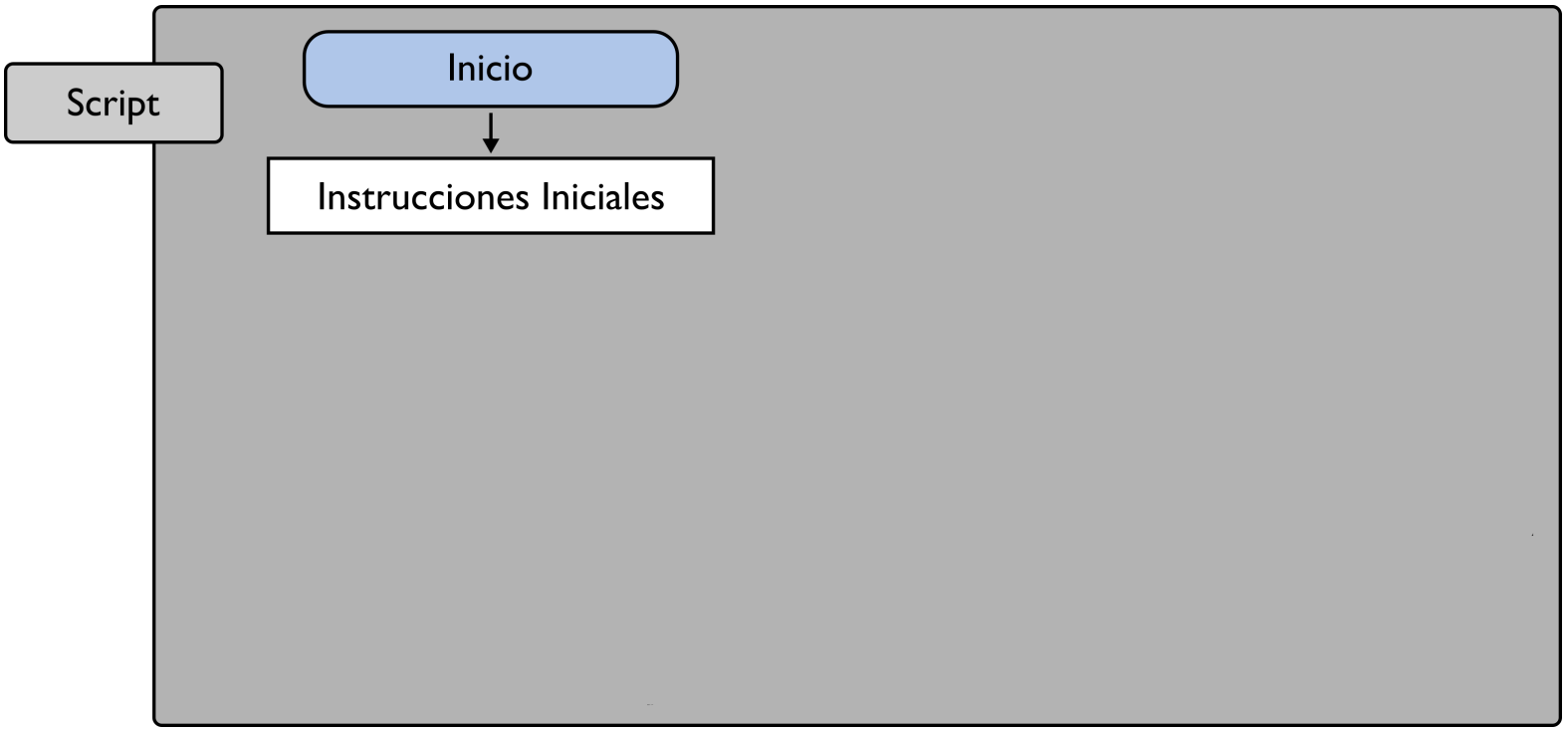
Límites de los programas secuenciales

- Ejemplo: Escribe un *programa*, que pida un número real y calcule su absoluto.

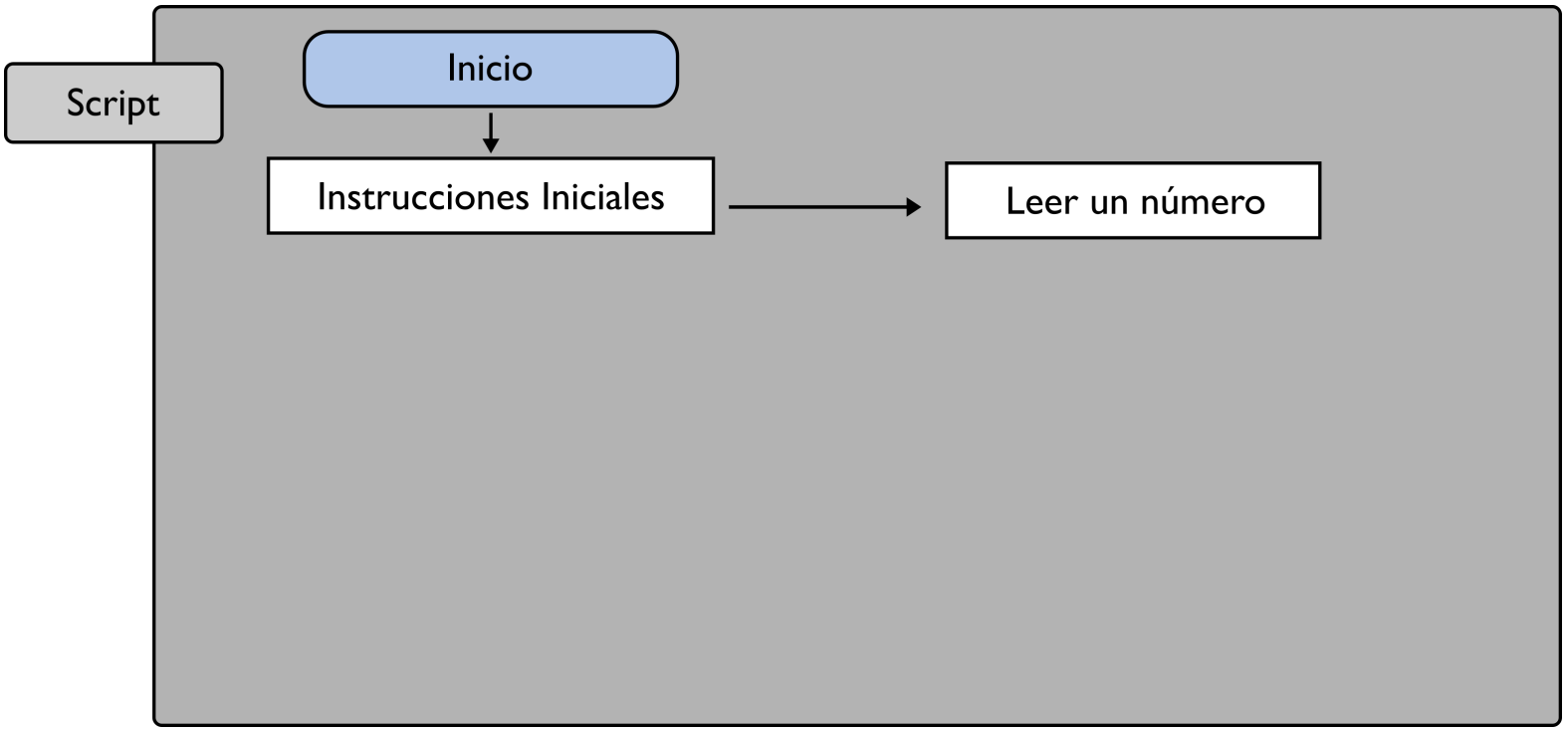
Límites de los programas secuenciales



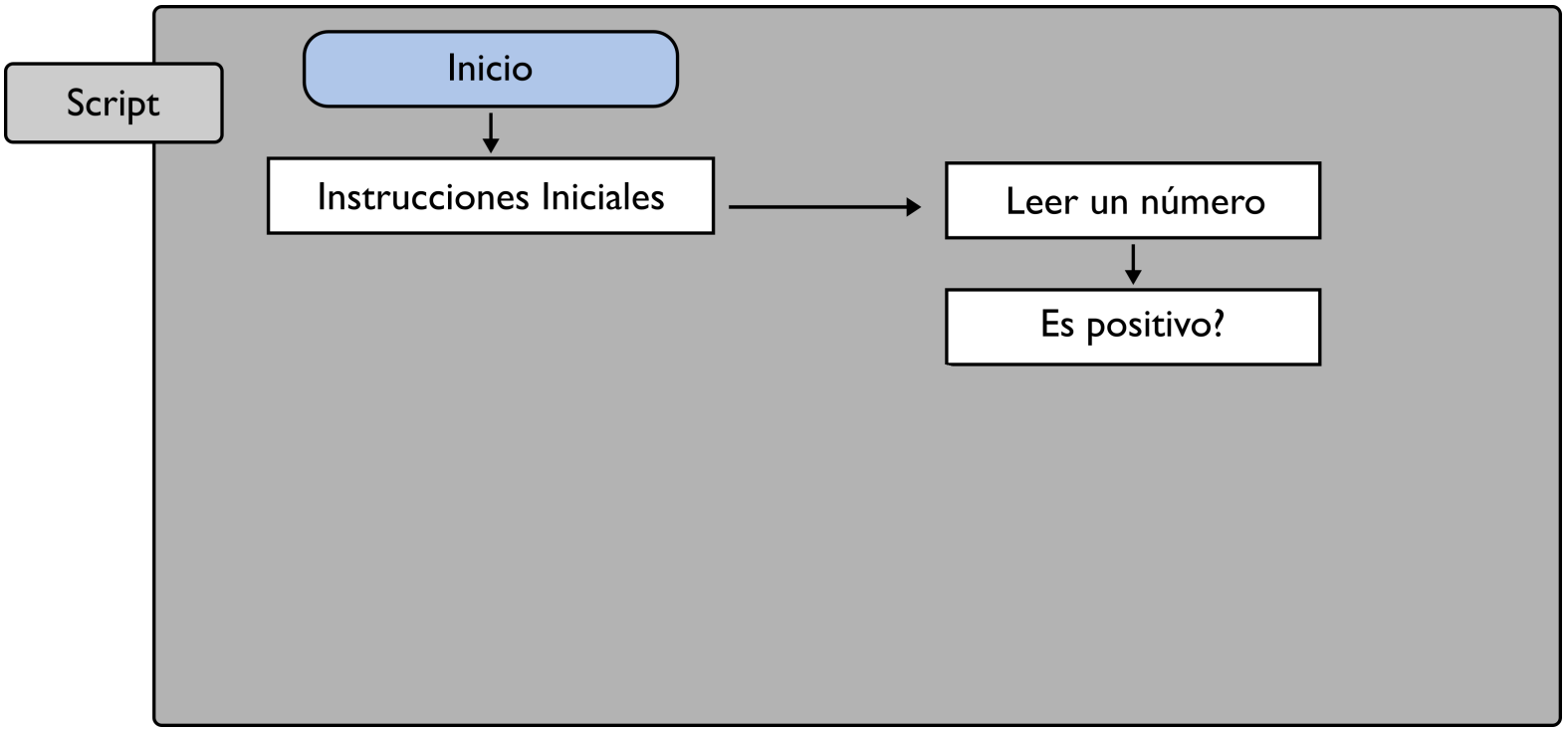
Límites de los programas secuenciales



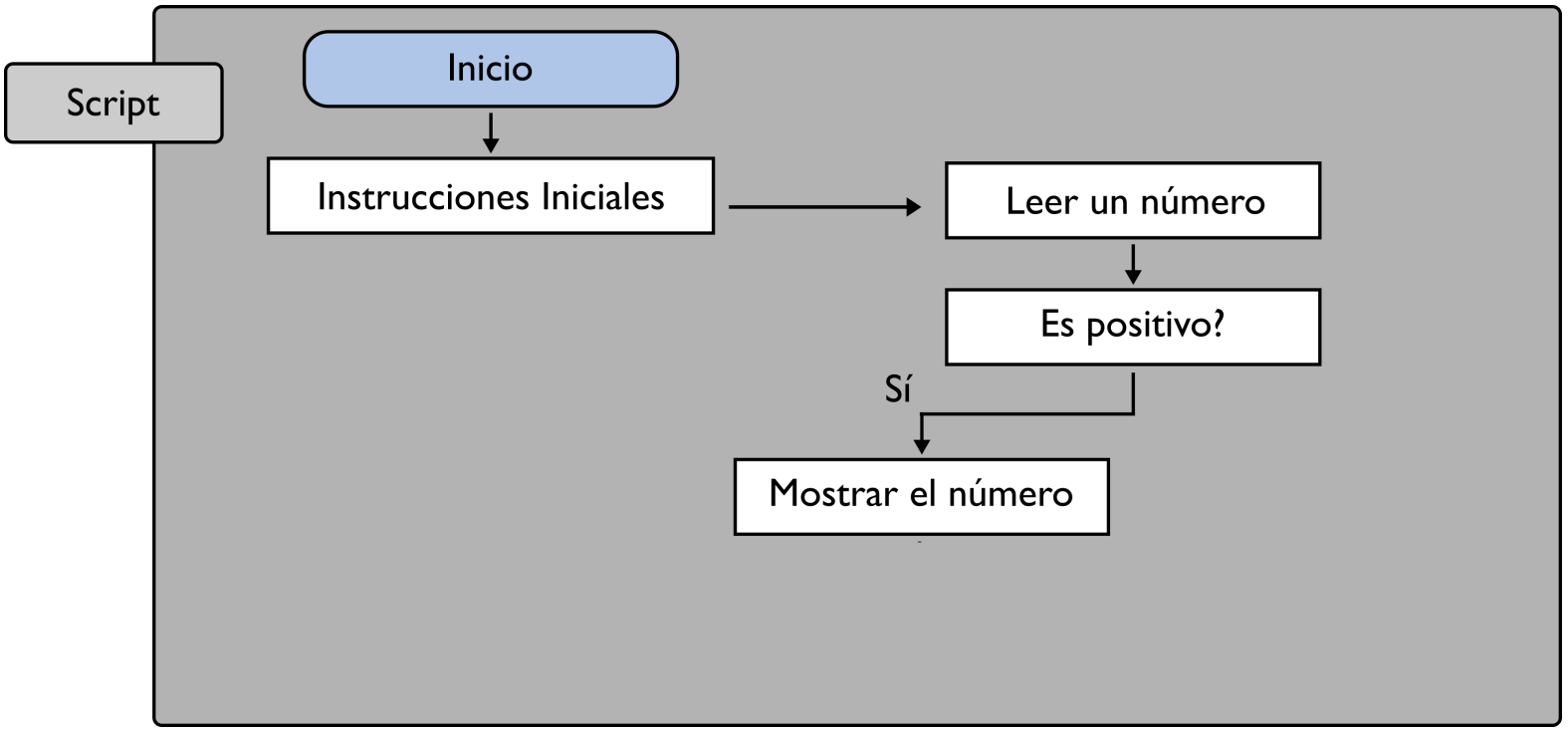
Límites de los programas secuenciales



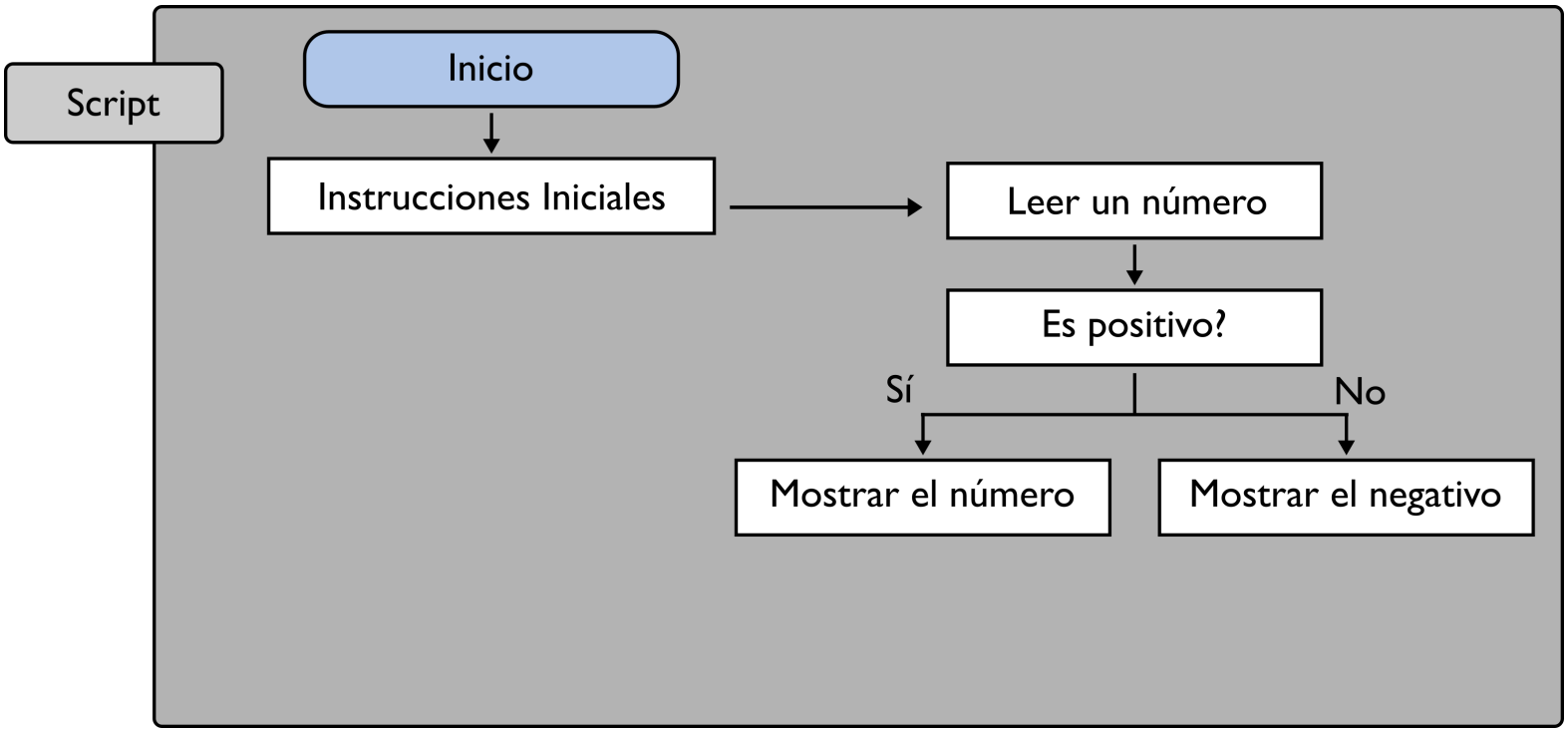
Límites de los programas secuenciales



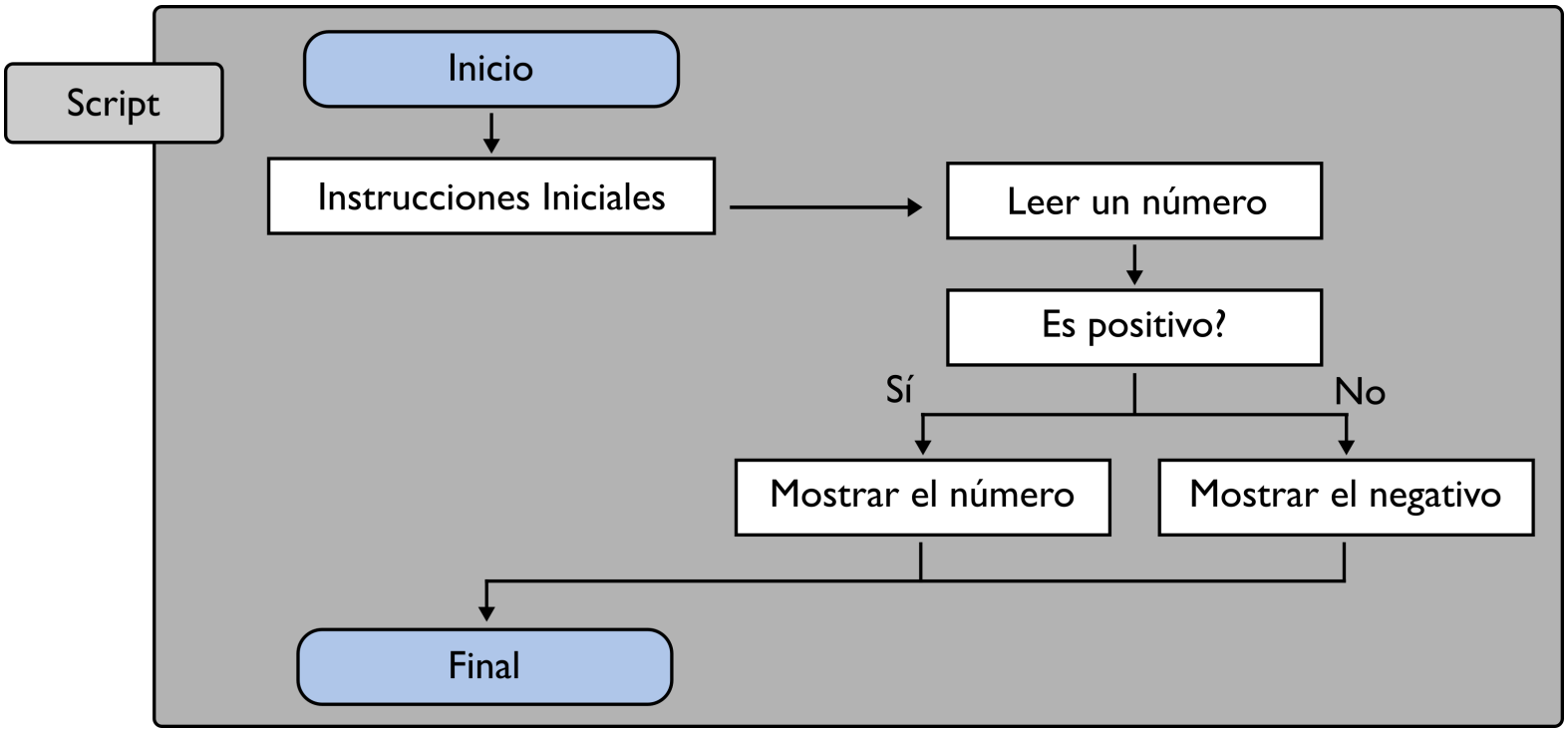
Límites de los programas secuenciales



Límites de los programas secuenciales



Límites de los programas secuenciales



Algoritmo

Ejemplo de calcular el valor absoluto

- El diagrama anterior puede representarse como algoritmo.
- El **Algoritmo** es la representación gráfica del proceso de programación.
- Algo intermedio del lenguaje de programación y el lenguaje natural.
- Sirve como Guía para visualizar el proceso de forma clara, ordenada y gráfica, facilitando la programación.

Algoritmo

- El algoritmo es un concepto básico para programar
- Ayudan a planificar lo que debe hacer el programa.
- Programar requiere dos competencias:
  a) Diseñar una solución específica y estructurada;
  b) La capacidad de traducir esta solución a otro lenguaje.

Estructura de control `if`

- La estructura
- `if` permite saltar una o un grupo de instrucciones.
- Permite tomar diferentes vías/acciones en la ejecución
- No tienen capacidad para repetir instrucciones o modificar el flujo del programa.

Estructura de control `if`

- En esta asignatura utilizaremos con 3 estructuras `if`:
  1. La estructura `if`;
  2. La estructura `if-else`;
  3. La estructrra `if-elif-else`.

Estructura `if`

Estructura `if`

- La estructura `if` permite definir si una instrucción debe ejecutarse o no

☞
```
if exprexion_logica:
    instrucciones
```

- Si se cumple `exprexion_logica`, se ejecutará `instrucciones`.
  - En caso contrario, no.

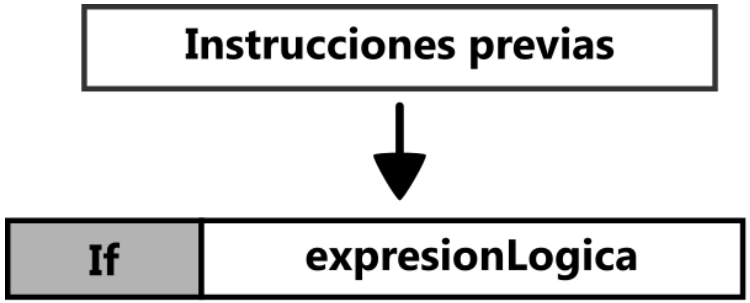
Estructura `if`

Instrucciones previas

☞
```
[instruciones
anteriores]
if exprexion_logica:
    instrucciones
[siguientes
instrucciones]
```

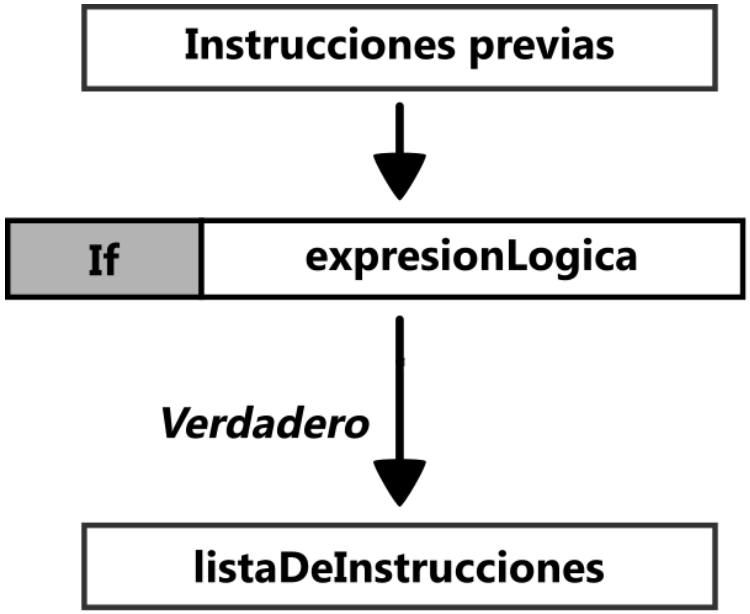
Estructura `if`

☞
```
[instruciones
anteriores]
if exprexion_logica:
    instrucciones
[siguientes
instrucciones]
```



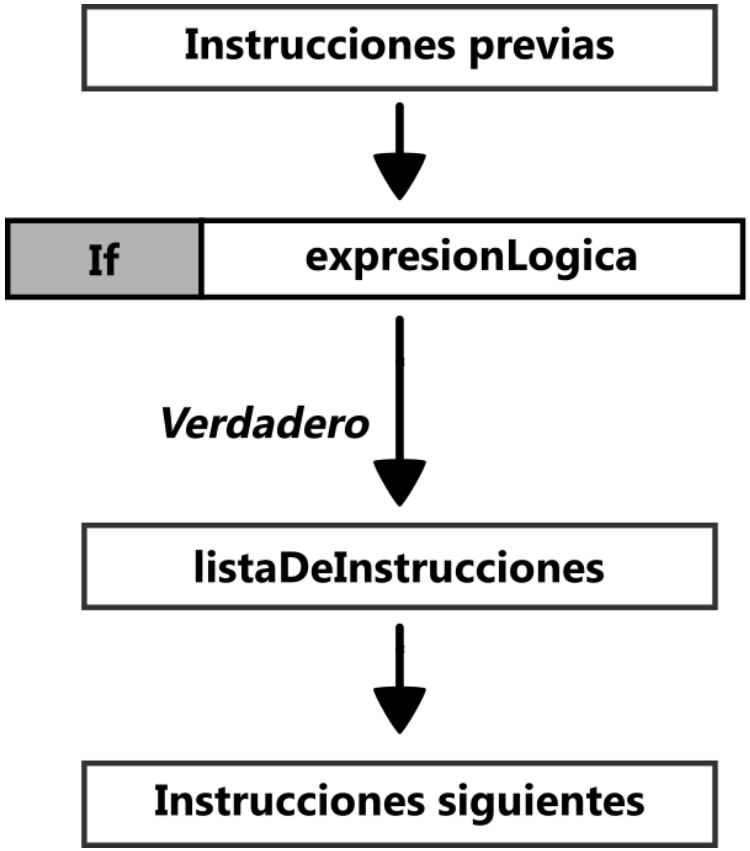
Estructura `if`

☞
```
[instrucciones
anteriores]
if exprexion_logica:
    instrucciones
[siguientes
instrucciones]
```



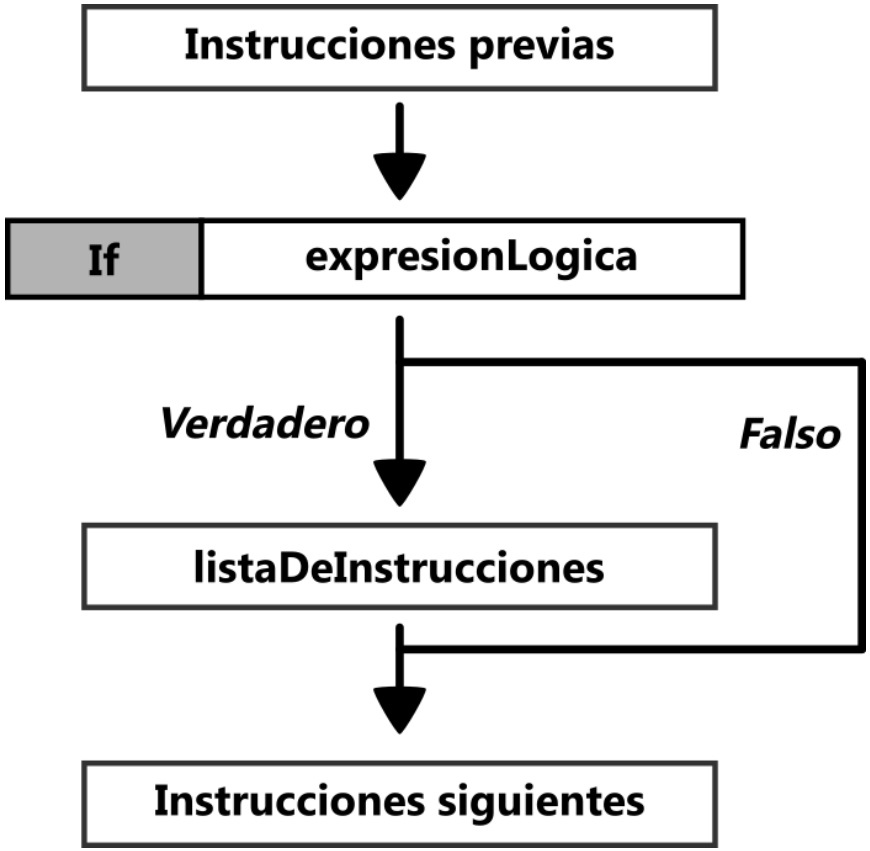
Estructura `if`

☞
```
[instrucciones
anteriores]
if exprexion_logica:
    instrucciones
[siguientes
instrucciones]
```



Estructura `if`

☞
```
[instrucciones
anteriores]
if exprexion_logica:
    instrucciones
[siguientes
instrucciones]
```



Estructura `if`

P: Entonces que puedo poner en `exprexion_logica`?

R: Un booleano o cualquier cosa que se pueda evaluar como, True/False.

R: Cualquier comparacióno función, combinación de funciones, ...

P: Entonces, que se puede incluir en `instrucciones`?

- Cualquier cosa, incluso más estructuras `if` si fuera necesario.
- Eso si, todas las líneas requieren estar indentadas para especificar que se incluyen dentro del `if`.

Número positibo, negatibo o 0.

- **Ejercicio** Crea un programa que indique si un número entero es positivo, negativo o 0.

¿Esta entre 0 y 10?.

- **Ejercicio** Crea un programa que indica si un número está entre 0 y 10.

Calcula el número absoluto.

- **Ejercicio** Define la función que calcula el absoluto de un número.

Estructura `if-else`

Estructura `if-else`

- `if-else` egitura instrukzio zerrenda bat edo bestea exekutatzea ahalbidetzen du

☞
```
if exprexion_logica:
    instrucciones_A
else:
    instrucciones_B
```

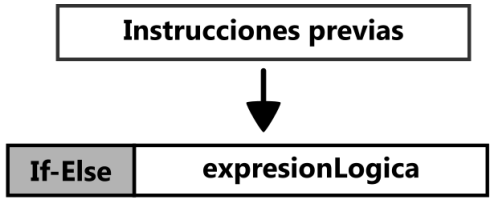
- Si se cumple `expresion_logica`, se ejecutará `instruccionesA`.
  - En caso contrario, se ejecutará `instrucciones_B`.

Estructura Estructura if-else

```
[instruciones
anteriores]
if exprexion_logica:
    instrucciones
else:
    instrucciones_B
[siguientes
instrucciones]
```

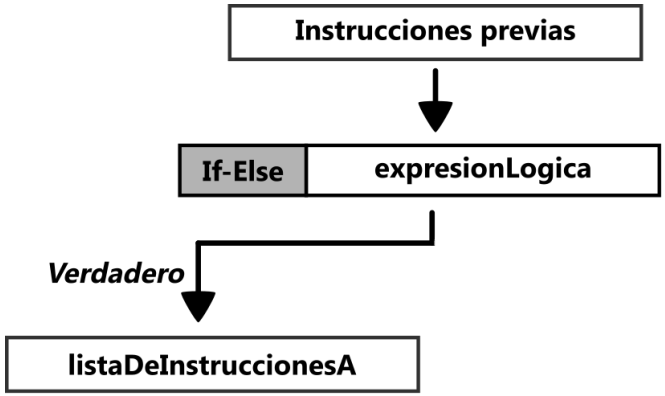
Estructura if-else

```
[instruciones
anteriores]
if exprexion_logica:
    instrucciones
else:
    instrucciones_B
[siguientes
instrucciones]
```



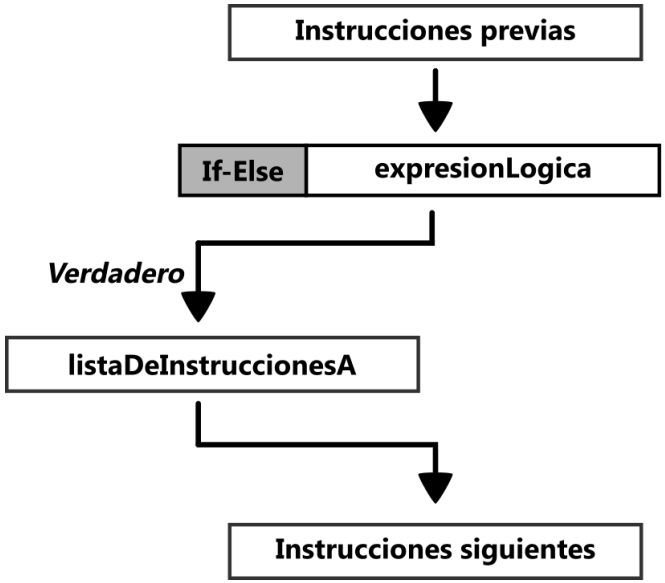
Estructura if-else

```
[instruciones anteriores]
if exprexion_logica:
    instrucciones
else:
    instrucciones_B
[siguientes instrucciones]
```



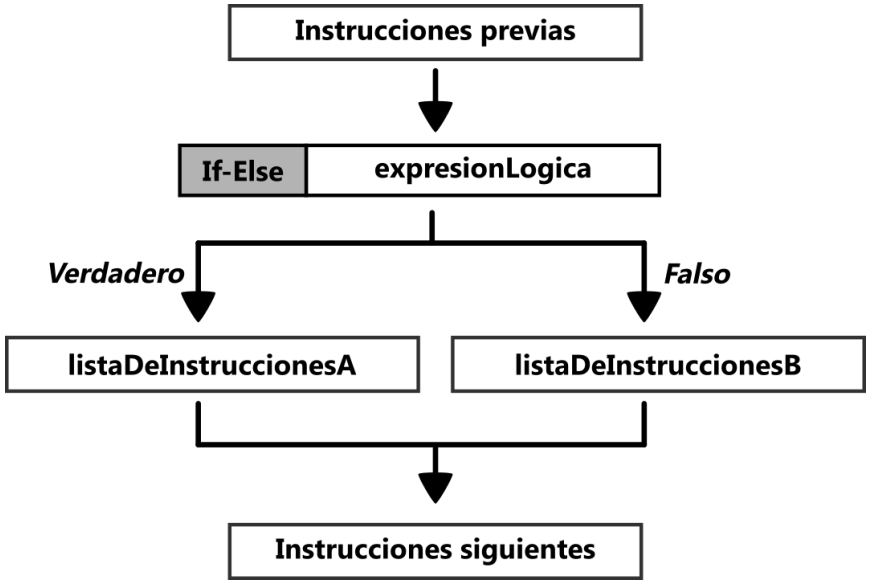
Estructura if-else

☞
```
[instruciones
anteriores]
if exprexion_logica:
    instrucciones
else:
    instrucciones_B
[siguientes
instrucciones]
```



Estructura if-else

☞
```
[instruciones
anteriores]
if exprexion_logica:
    instrucciones
else:
    instrucciones_B
[siguientes
instrucciones]
```



Estructura `if-else`

P: Entonces que puedo poner en `exprexion_logica`?

R: Un booleano o cualquier cosa que se pueda evaluar como, True/False.

R: Cualquier comparacióno función, combinación de funciones, ...

P: Entonces, que se puede incluir en `instrucciones_A` o `instrucciones_B`?

- Cualquier cosa, incluso más estructuras `if` o `if-else` si fuera necesario.
- Eso si, todas las líneas requieren estar indentadas para especificar que se incluyen dentro de cualquier estructura `if`.

Número positibo, negatibo o 0.

- **Ejercicio** Crea un programa que indique si un número entero es positivo, negativo o 0. Utiliza la estructura `if-else`.

¿Esta entre 0 y 10?.

- **Ejercicio** Crea un programa que indica si un número está entre 0 y 10. Utiliza la estructura `if-else`.

Estructuración de programas

- Los programas que creamos son secuenciales
- Se ejecutan de arriba a abajo.
- Se ejecutan todas las líneas.
- Normalmente, cuando los programas crecen, el uso de estructuras estándar ayuda a la comprensión
- La estructuración estándar se basa en bloques, y se suelen utilizar 3 bloques
  1. Bloque de variables: al principio del archivo se asignan casi todas las variables
  2. Bloque de la lógica del programa
  3. Bloque de presentación de resultados

Funciones

Encapsulación

- En los grandes programas la lectura y escritura se complica.
- El encapsulado de las instrucciones es interesante para ganar un punto de abstracción.
- Las funciones que encapsulan más de una instrucción, algunos ejemplos son: `max ()`,`min ()`,`abs ()`, ...
  - Dentro de una función se ejecutan grupos de instrucciones.
  - Permiten olvidarse del código que hay dentro.
  - Hacen que el programa sea fácilmente entendible y permiten distribuir el código.

Funtzioak

- En informática, una subrutina o subprograma (también llamado procedimiento, función o rutina) puede presentarse como una idea general como un subalgoritmo que forma parte del algoritmo principal.
- Cada función es un subprograma que permite resolver una tarea concreta.
- Algunos lenguajes de programación, Basic .net o Fortran, por ejemplo, llamaron función a las subrutinas que devuelven un valor.
- Hoy en día, la palabra función puede considerarse estándar.

Funtzioak

- Son bloques de código que se pueden ejecutar como unidad funcional.
- Desde cualquier sección se puede llamar o ejecutar tantas veces como se quiera.
- Puedes recibir los valores en la opción.
- Se ejecuta y puede devolver un valor o varios.
- Desde el punto de vista organizativo, podemos decir que una función es algo que permite un determinado orden en una mezcla de algoritmos.

Funciones: sintaxis en python

```
def un_nombre(arg1, arg2, ...):
    [Lista de intrucciones]
```

- Donde la palabra reservada `def` el nombre y argumentos de la nueva función,
- `un_nombre` el nombre que quieras,
- `arg1`, `arg2`,... tantos argumentos como desees,
- `[Lista de intrucciones]`, tantas instrucciones como desees, todas ellas indetadas.

Operaciones entre variables

- **Ejercicio** Escribe la función que indica el resultado de multiplicar dos números.

Operaciones entre variables: resultado

- ¿Se pueden asignar dos variables en una línea?
- ¿Este Código define los tres bloques de estructuras?
- ¿Cómo definiríamos una función que genera multiplicaciones?

```
1 # python script
2 multiplicando, multiplicador = 5, 2
3
4 producto = multiplicando*multiplicador
5
6 print('El producto es',producto)
```

Código: Ariketaren emaitza.

Operaciones entre variables: resultado en función

- La función siempre se define al inicio del programa
- Una vez definido se puede ejecutar tantas veces como se desee
- Las instrucciones dentro de la función deben indicarse siempre indentadas (con una tabulación)

```
1 # definir la funcion
2 def mutiplicacion(mcando,mcador):
3     producto = mcando*mcador
4     print('El producto es',producto)
5
6 # 3 multiplicaciones
7 mutiplicacion(2, 2)
8 mutiplicacion(4, 2)
9 mutiplicacion(5, 2)
```

Código: Resultado del ejercicio.

Explicación de la ejecución de la función

```
1    multiplicacion(2, 2)
```

Código: Ejemplo de ejecución.

- Cuando se ejecuta `multiplicación(2,3)` el programa hace lo siguiente
- Asigna los valores 2 y 3 en orden con los nombre definidos como argumentos y se ejecutan las líneas dentro de la función con esos valores

```
1 mcando,mcador = 2,3
2 producto = mcando*mcador
3 print('El producto es',producto)
```

Código: Ejemplo de ejecución.

Funciones

- Ten en cuenta que a la hora de programar las funciones deben definirse en las primeras líneas
- Los argumentos definidos en la función no tienen por qué asignarsele valor, se asignará al ejecutarlos
- Las instrucciones que están dentro de la función deben indicarse siempre indentadas
- Las variables definidas dentro de la función, sólo existen dentro de la función
- Entonces, ¿qué devuelven las funciones?
  - Si no se dice lo contrario y por defecto, una función devuelve siempre `None`
  - Pero, ¿cómo podemos definir la devolución?

Funtzioak: sintaxia pythonen

```
def un_nombre(arg1, arg2, ...):
    [Lista de intrucciones]
    return variable
```

- Donde la palabra reservada `def` el nombre y argumentos de la nueva función,
- `un_nombre` el nombre que quieras,
- `arg1`, `arg2`,... tantos argumentos como desees,
- `[Lista de intrucciones]`, tantas instrucciones como desees, todas ellas indetadas.
- `return` palabra reservada que devuelve un valor fuera de la función
- `variable` valor que devolverá la función.

Función de multiplicación

- **Ejercicio** Escribe un programa que indique el resultado de multiplicar dos números. En esta ocasión, los valores calculados quedarán fuera de la función. Sigue las reglas de estructura para programar: primero las funciones, luego la asignación de variables, a continuación la lógica del programa y, por último, los comandos `print`.

Operaciones entre variables: función definida

- Qué está mal en el siguiente código?

```
1  # definir funciones
2  def multiplicacion(multiplicando,multiplicador):
3      producto = multiplicando*multiplicador
4      return producto
5
6  # 3 multiplicaciones seguidas
7  multiplicacion(2,2)
8  print('El producto es',producto)
9  multiplicacion(4,2)
10 print('El producto es',producto)
11 multiplicacion(5,2)
12 print('El producto es',producto)
```

Código: Ariketaren emaitza.

Operaciones entre variables: función definida

- Las variables tienen un ámbito de variable, es decir, un lugar y tiempo durante la ejecución en el que existen.
- Por ejemplo, si asignamos una variable dentro de una función, sólo podremos utilizarla dentro de esa función.

```
1 def definir_PI():
2     PI=3.1416
3
4 print(PI*5)
```

Código: Una variable dentro de una función.

- ¿Funcionaría, entonces, este código?

Operaciones entre variables: función definida

- Las variables definidas en el programa principal, si se han definido antes que la función, seguirán vivas dentro de la función.

```
1 PI = 4.1416
2 def pantailaratuPI():
3         print(PI)
4
5 pantailaratuPI()
```

Código: Definiendo una variable global.

- ¿Funcionaría, entonces, este código?
- Aun funcionando, definir una variable global para que exista dentro de una función es una mala práctica de programación.

Funciones: Devolver más de un valor

```
def cualquier_nombre(arg1, arg2, ...):
    [Lista de instrucciones]
    return val1, val2
```

- Donde `def` es la palabra clave que indica la definición de una nueva función,
- `cualquier_nombre` es el nombre de la función que desees,
- `arg1`, `arg2`,... son los parámetros de entrada que desees,
- `[Lista de instrucciones]`, es la lista de instrucciones, todas indentadas.
- La palabra clave `return` se usa para devolver los valores/variables definidos,
- `valores` son los valores que la función devolverá, separados por comas.

Funtzioak: devolviendo más de un valor

```
def un_nombre(arg1, arg2, ...):
    [Lista de intrucciones]
    return var1,var2
```

- Donde la palabra reservada `def` el nombre y argumentos de la nueva función,
- `un_nombre` el nombre que quieras,
- `arg1`, `arg2`,... tantos argumentos como desees,
- `[Lista de intrucciones]`, tantas instrucciones como desees, todas ellas indetadas.
- `return` palabra reservada que devuelve un valor fuera de la función
- `var1`, `var2` más de un valor separado por coma que devolverá la función.

Operaciones entre variables resultado

- El resultado de la división entre números enteros requiere dos variables...

```
1 # python script
2 dividendo, divisor = 5, 2
3
4 cociente = dividendo//divisor
5 resto = dividendo%divisor
6
7 print('El cociente es',cociente,'y el resto es',resto)
```

Código: Programa que calcula la división entre números enteros.

Operaciones entre variables: función definida

- Pero, ¿si las funciones`print` deben estar fuera del código principal?

```
1  # definicion de funcionesk
2  def division_entera(dividendo, divisor):
3      cociente = dividendo//divisor
4      resto = dividendo%divisor
5      print('El cociente es',cociente,'y el resto es',
           resto)
6
7  # tres divisiones
8  division_entera(2, 2)
9  division_entera(4, 2)
10 division_entera(5, 2)
```

Código: resultado del ejercicio.

Aldagaien arteko eragiketak: funtzioa definituta

```
1  # definicion de funcionesk
2  def division_entera(dividendo, divisor):
3      cociente = dividendo//divisor
4      resto = dividendo%divisor
5      return cociente,resto
6
7  # bi zatiketa
8  cociente,resto = division_entera(2, 2)
9  print('El cociente es',cociente,'y el resto es',resto)
10 cociente,resto = division_entera(4, 2)
11 print('El cociente es',cociente,'y el resto es',resto)
```

Código: Función de división devolviendo dos valores.

Funciones: Ejercicio para los alumnos

- Define las funciones necesarias para que el código siguiente funcione correctamente.

```
1 # definicion de funcionesk
2 ...
3 # Hiru zatiketa
4 print(division(4, 2))
5 print(suma(5, 2))
6 print(raiz_cuadrada(5))
7 print(potencia(5,2))
```

Código: Resultado del ejercicio.

Resumen

Resumen de contenidos

- Estructura de los programas
- Estructuras de control
  1. `if` egitura;
  2. `if-else` egitura;
- Funciones, capsulación y ámbito de la variable
- Ejercicios.