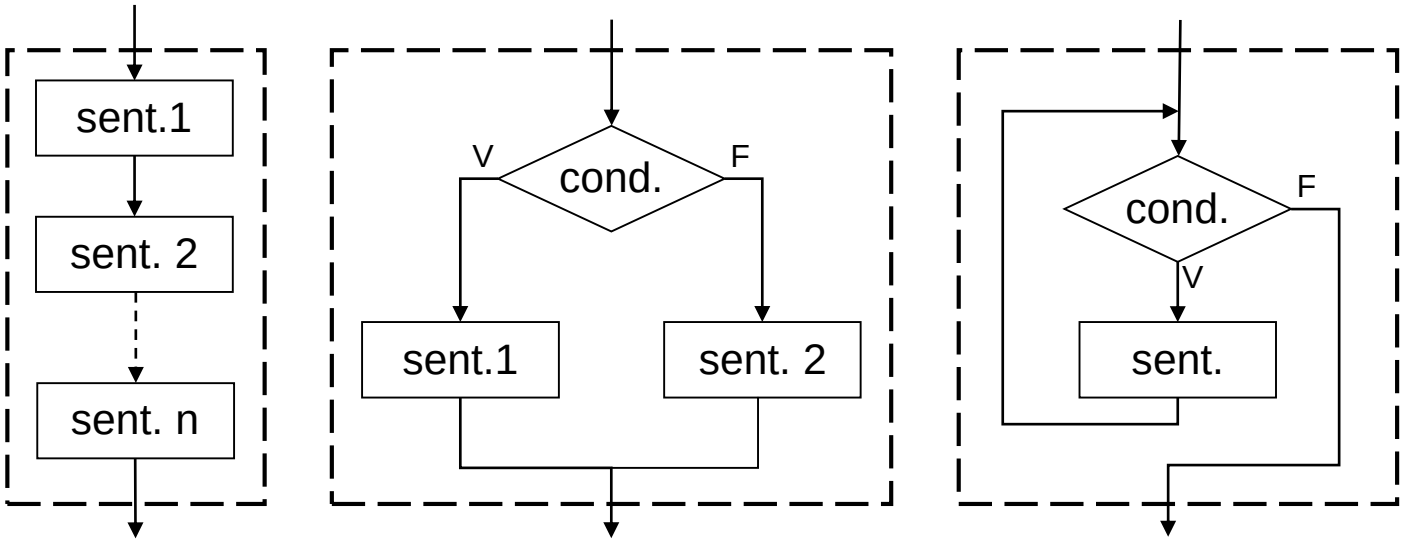
Tema 2. Introducción a la programación

- 2.1 Abstracción de problemas para su programación. Conceptos fundamentales
- 2.2 Variables, expresiones, asignación
- 2.3 Uso de entrada/salida por consola
- **2.4 Manejo de estructuras básicas de control de flujo: secuencial, alternativa y repetitiva**
- 2.5 Definición y uso de subprogramas y funciones. Ámbito de variables
- 2.6 Entrada/salida a ficheros
- 2.7 Tipos y estructuras de datos básicas: arrays

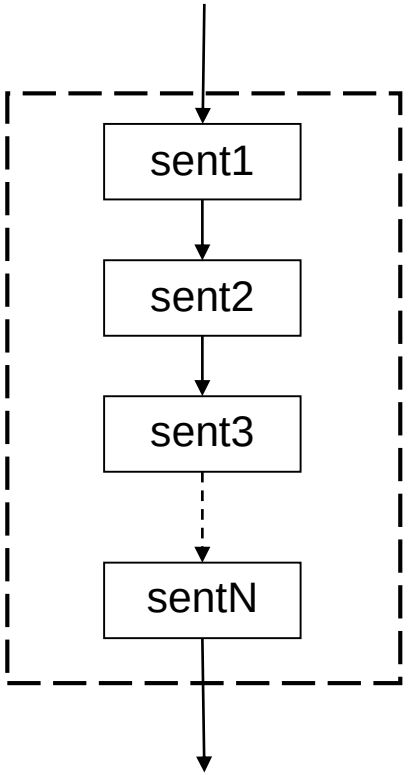
2.4 Estructuras de control fundamentales

- Existen 3 estructuras de control fundamentales:
  - Secuencial (**`BLOQUE`**)
  - Aternativa (**`SI-ENTONCES-SI_NO`**)
  - Repetitiva (**`MIENTRAS`**)



`BLOQUE` **`SI-ENTONCES-SI_NO`** **`MIENTRAS`**

Estructura secuencial



- Consiste en ejecutar una sentencia a continuación de otra.

Implementación

- Se pueden agrupar varias sentencias para formar **una única sentencia,** denominada "*sentencia compuesta*".
- En Python, Lo que "delimita" el bloque es que todas tengan el mismo nivel de *indentación*, es decir, el mismo número de espacios por la izda.

```
sent1
sent2
sent3
  ...
sentN
```

Estructura secuencial

- Se puede usar “punto y coma” para separar sentencias si están en la misma línea.
- Si están en líneas separadas no es necesario, pero hay que prestar atención a la “indentación”

Ejemplos:

Correcto

```
sent1
sent2
sent3
sent4
Sent5
```

Incorrecto

```
sent1
 sent2
sent3
sent4
sent5
```

Incorrecto

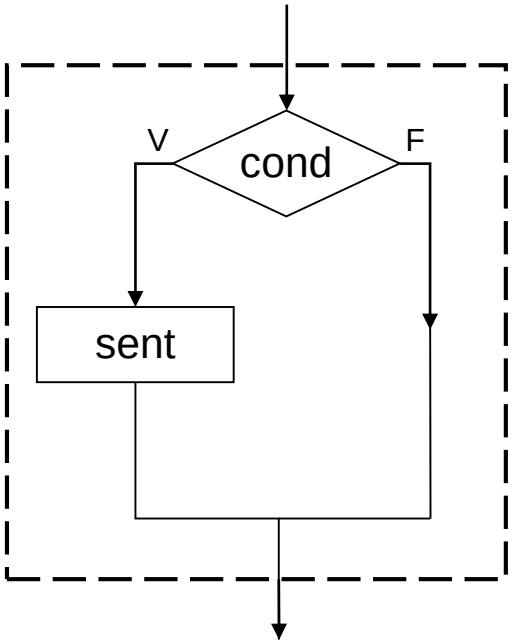
```
 sent1
sent2
 sent3
 sent4
 sent5
```

Estructura alternativa

- Permite elegir entre dos alternativas, según sea *verdadera* o *falsa*, la condición que se evalúa.

- Existen dos subtipos
  - Aternativa simple (`if`)
  - Aternativa doble (`if-else`)

Estructura alternativa simple (`if`)



- Si la "condición" es *verdadera* se ejecuta la "sentencia". Si no, no se ejecuta ninguna acción.

Implementación

```
if cond:
    sent
```

Estructura alternativa simple (`if`)

- En el caso de que haya más de una “sentencia”, se deberá formar una única *sentencia compuesta* usando el mismo nivel de indentación.

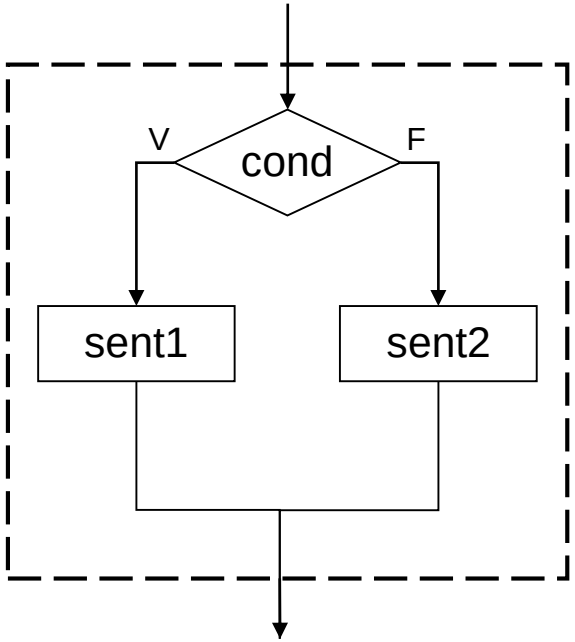
Ejemplo

Suponiendo que la variable **`nota`** contiene la calificación de un alumno, el siguiente fragmento de programa determina si ha superado la prueba.

```
# El programa obtiene la nota por algún medio

if nota >= 5.0:
    print(" Prueba superada “)
```

Estructura alternativa doble (`if-else`)



- Si la "condición" es *verdadera* se ejecuta la "sentencia1". Si no, se ejecuta la "sentencia2".

Implementación

```
if cond:
    sent1
else:
    sent2
```

- Nótese que el if y el else van alineados.

Estructura alternativa doble (`if-else`)

- Tanto “sent1” como “sent 2” deben de ser una única sentencia.

  Si hubiera más de una “sentencia”, se deberá formar una única *sentencia compuesta* usando el mismo nivel de indentación.

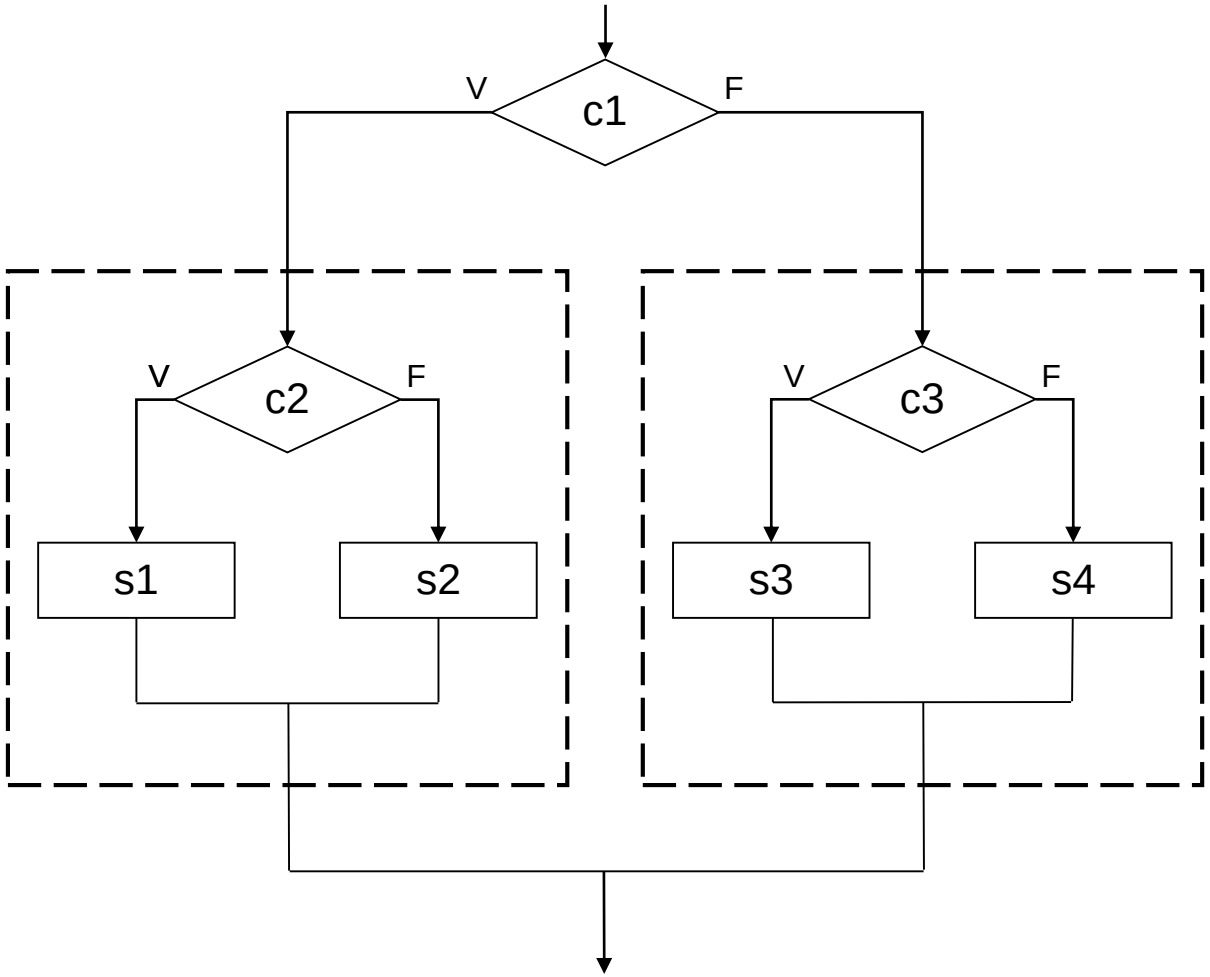
Ejemplo

El siguiente fragmento de programa determina si la variable **`num`** de tipo entero contiene un número par o impar.

```
# El programa inicializa num por algún medio

if num%2 == 0:
    print(" Es par ")
else:
    print(" Es impar ")
```

Anidamiento de estructuras alternativas



Implementación

```
if c1:
    if c2:
        s1
    else:
        s2
else:
    if c3:
        s3
    else:
        s4
```

Estructura multialternativa (`if-elif-else`)

- Podemos verla como una serie de sentencias `if-else` anidadas.
- Permite elegir entre varias alternativas:

```
if cond1:
      sent1
elif cond2:
      sent2
elif cond3:
      sent3
...
elif condN:
      sentN
else:
      sentencia     #cualquier otro caso no contemplado
```

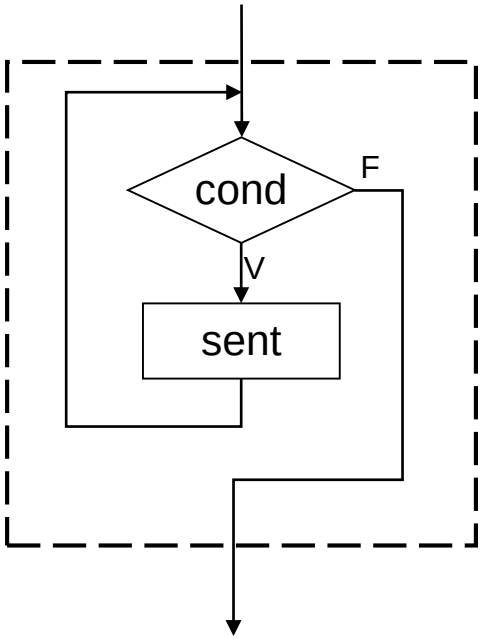
Estructura multialternativa (if-elif-else)

Ejemplo

```
nota = int(input("Introduzca la nota del examen: "))

if (nota >= 0) and (nota < 5):
     print(" Suspenso ")
elif (nota >= 5) and (nota < 7):
     print(" Aprobado ")
elif (nota >= 7) and (nota < 9):
     print(" Notable ")
elif (nota >= 9) and (nota <= 10):
     print(" Sobresaliente ")
else:
     print(" Nota no válida ")
```

Estructura repetitiva (bucle mientras)



- Ejecución repetitiva de un conjunto de sentencias mientras la evaluación de una condición sea verdadera
- La “condición” se evalúa al principio.
- Mientras la “condición” sea *verdadera* se ejecuta las sentencias (“sent”).

Implementación

```
while cond:
    sent
```

Estructura repetitiva (bucle mientras)

- Si hubiera más de una "sentencia" dentro del bucle, se deberá formar una única *sentencia compuesta* usando el mismo nivel de indentación, como se ve en el siguiente ejemplo.

Ejemplo

El siguiente bucle calcula el número de dígitos de un entero `n` dado (introducido por teclado).

```
n = int(input("Numero entero: "))
a = n
digitos = 1
while a//10 != 0:
    digitos = digitos + 1
    a = a // 10

print(n, "tiene", digitos, "digitos")
```

Estructura repetitiva (bucle mientras)

Cómo construir un bucle `while`

- Si bien cada programa con un bucle contiene un conjunto de variables y una secuencia de instrucciones diferente según el problema a resolver, es posible establecer algunas reglas sobre cómo construir un bucle (`while`) que se pueden y deben aplicar en todos los casos.

- Para obtener las reglas de construcción del bucle, es necesario ver cómo es la secuencia de **estados de las variables** que se genera en su ejecución paso a paso

  Estado de las variables (o estado del programa): es el valor de las variables del programa en cada paso.

Estructura repetitiva (bucle mientras)

```
n = int(input("Numero entero: "))
a = n
digitos = 1                # bloque → estado inicial
while a//10 != 0:
    digitos = digitos + 1
    a = a // 10            # bloque → cambio de estado
```

Ejecución paso a paso

Entrada:
n = 62574

Solución del problema

Estado inicial

Cambio de estado

Estado final

Estado	Variables			*a//10≠0*
	n	*digitos*	*a*	
S_0	62574	1	62574	V
S_1	62574	2	6257	V
S_2	62574	3	625	V
S_3	62574	4	62	V
S_4	62574	5	6	F

Estructura repetitiva (bucle mientras)

```
# Estado inicial
Inicializar_variables
while ¬ Condición_parada:
    # Cambio de estado
    Sentencia_interna
```

Ejecución paso a paso

Secuencia de estados

Solución del problema

Paso 3. Construir el bucle

Paso 1

Buscar unos pocos elementos iniciales de la secuencia de estados para el problema a resolver (los suficientes)

Paso 2

Deducir:

- Estado inicial
- Condición de parada del bucle
- Cambio de estado

Estructura repetitiva (bucle mientras)

Reglas de construcción del bucle (del Paso 2 al Paso 3)

```
# Regla 1. Establecer el estado inicial
Inicializar_variables    # con datos de entrada o asignaciones
# Regla 2. Determinar la condición de parada
while ¬ Condición_parada:
    # Regla 3. Realizar el cambio de estado
    Sentencia_interna    # bloque de sentencias necesario para
                         # cambiar el estado de cada variable
```

Estado final

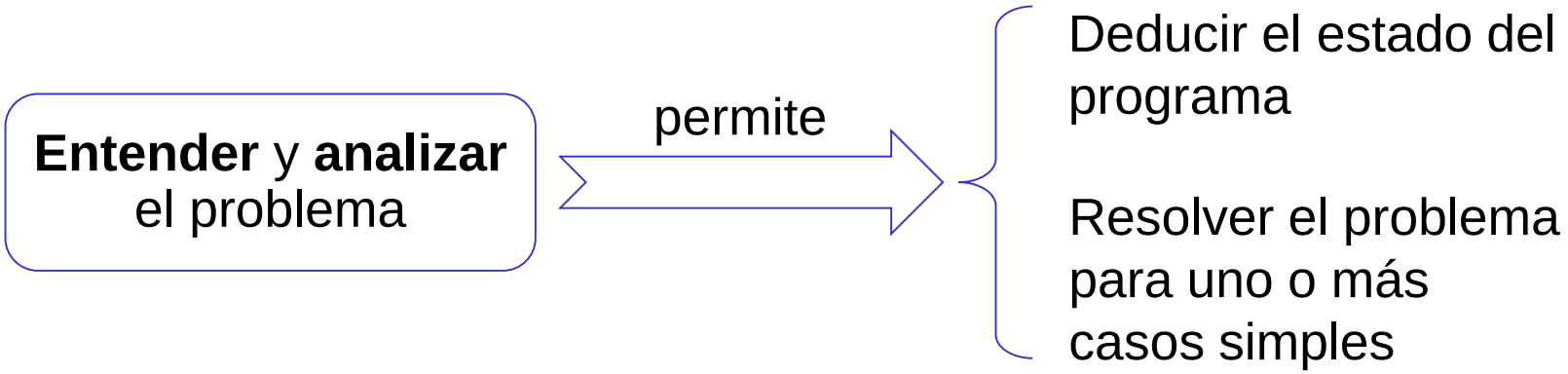
Para evitar errores:

De las variables que componen el estado del programa, aquellas que contienen datos de entrada al programa no deberían modificarse.

- Así, para el programa de ejemplo, la variable `n` que contiene el número entero proporcionado no se modifica.

Estructura repetitiva (bucle mientras)

- Sólo resta saber como obtener la secuencia de estados inicial (Paso 1):



- En lo que se refiere al estado del programa siempre se puede tener en cuenta lo siguiente:

> El estado de un programa con un bucle está constituido por la información (o datos almacenados en variables) necesaria para poder realizar el cambio de estado.

Esto se aplicará en el ejemplo que se mostrará a continuación.

Estructura repetitiva (bucle mientras)

Ejemplo. Escribir un programa que permita obtener en pantalla la secuencia de enteros: `1`, `3`, `6`, `10`, `15`, … que no superen un valor `umbral` (≥`0`) dado.

No debería ser un problema saber quién es el siguiente término, el `21`, y si esto es así, es porque se habrá determinando cómo pasar de un término al siguiente: `1`, `1+2=3`, `3+3=6`, `6+4=10`, `10+5=15`, …

Estado	*umbral*	*termino*	*cantidad*
S_0	20	1	2
S_1	20	1+2=3	3
S_2	20	3+3=6	4
S_3	20	6+4=10	5
S_4	20	10+5=15	6
S_5	20	15+6=21	7

1. *Estado inicial*:
```
umbral=int(input(
"Umbral: "))
termino=1
cantidad=2
```

2. *Condición de parada*:
```
termino>umbral
```

Estructura repetitiva (bucle mientras)

3. *Cambio de estado*:

```
print(termino,end=“ “)
termino=termino+cantidad
cantidad=cantidad+1
```

- Para cambiar el estado hay que cambiar cada una de las variables según se deduzca de la tabla.

```
# establecer el estado inicial
umbral=int(input("Umbral: "))
termino=1
cantidad=2
# while ¬ Condición_parada:
while termino <= umbral:
    # cambiar el estado
    print(termino, end=“ “)
    termino=termino+cantidad
    cantidad=cantidad+1
print()
```

Estructura repetitiva (bucle controlado por contador)

- Este tipo de bucle se utiliza para repetir una acción un número predeterminado de veces.

- Forma básica:

```
for vc in secuencia_de_valores:
    sent
```

- En estos casos existe una variable de control (*vc*) del bucle, que va tomando sucesivamente los valores de la secuencia
- Los valores de la secuencia se generarán con `range()`

Bucle controlado por contador: `range()`

`range()`

- Genera una secuencia de números enteros.
- Lo vamos a utilizar con 3 parámetros:
  - **`range(m, n, p)`**
    - El parámetro “p” es el “paso” o incremento entre los valores generados. Puede ser positivo o negativo.
    - Si el paso es positivo, tendríamos:
      - Si m<n, crea una lista creciente de enteros, comprendidos entre “m” y el anterior a “n” , espaciados por un paso “p”.
      - Si m>=n crearía una secuencia vacía.
    - Si el paso es negativo, la secuencia sería decreciente y m debería de ser mayor que n para que se generara una secuencia no vacía.

Bucle controlado por contador: `range()`

Ejemplos

`range(2, 10, 2)` genera la secuencia: `2, 4, 6, 8`

`range(10, 4, -2)` genera la secuencia : `10, 8, 6`

Bucle for

- La *variable de control* "vc" se inicializa a "ini".
- Al final de cada iteración se va modificando en el valor indicado por "p".
- Finaliza al alcanzar o rebasar el valor de "fin"

Implementación

```
for vc in range(ini, fin+1, p):
    sent
```

Bucle for

- En el caso de que haya más de una sentencia, se deberá formar una única *sentencia compuesta* usando el mismo nivel de indentación.

Ejemplo

El siguiente bucle imprime los números del 1 y 10 (uno en cada línea), los suma e imprime su suma

```
suma = 0
for i in range(1, 11, 1):
    print(i)
    suma = suma + i
print(suma)
```

Bucle for

Ejemplo

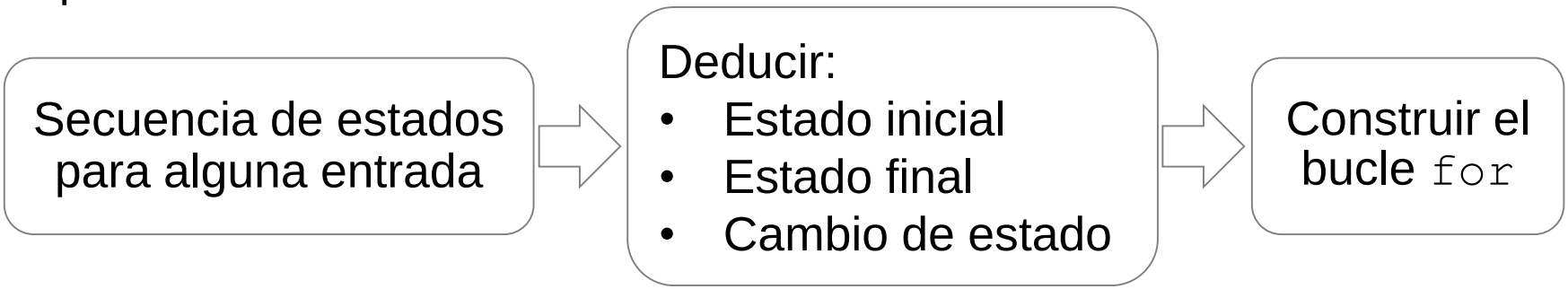
El siguiente bucle escribe los números comprendidos entre 10 y 1, ambos incluidos, uno en cada línea, en orden descendente.

```
for i in range(10, 0, -1):
    print(i)
```

Estructura repetitiva (bucle for)

Cómo construir un bucle `for`

- Para construir un bucle `for` se procederá de forma análoga a la ya vista para el `while`.



- Así, las reglas de construcción del bucle `for` serán similares a las dadas para el bucle `while`. Las únicas diferencias se deben a las características propias de este tipo de bucle:
  1. El bucle `for` sólo se puede utilizar si se conoce a priori el número de iteraciones a realizar (o el estado inicial y final de la variable de control). Son estos valores los que indican cuando ha de terminar el bucle.

Estructura repetitiva (bucle for)

2. La inicialización de las variables y su cambio de estado se hace igual que para el bucle `while`, excepto para la variable de control del bucle.
   - La variable de control del bucle, `vc`, se inicializa en el `range` y el *paso* de éste establece su cambio de estado. Además, esta es la primera variable que cambia de estado en cada iteración.

Reglas de construcción del bucle

```
# Regla 1. Establecer el estado inicial, excepto para vc
Inicializar_variables   # con datos de entrada o asignaciones

# Regla 2. Establecer el rango del for
for vc in range(inicial, final, paso):
# Regla 3. Realizar el cambio de estado
    Sentencia_interna   # bloque de sentencias necesario para
                        # cambiar el estado de cada variable,
                        # excepto para vc
```

`range` ← *Estado inicial y final o número de iteraciones*

Estructura repetitiva (bucle for)

Ejemplo. Dado un entero `n` (≥`0`), escribir un programa que permita obtener en pantalla los `n` primeros términos de la secuencia: `1`, `3`, `6`, `10`, `15`, …

El problema ya se trató cuando se vio el bucle `while`, así que se partirá de una tabla de estados similar cambiando la entrada, que aquí es el valor de `n`.

En particular, se muestra la tabla correspondiente a la entrada `n=5`. Como variable de control se utilizará `cantidad`, en lugar de introducir una nueva.

Estado	*n*	*termino*	*cantidad*
S_0	5	1	1
S_1	5	1+2=3	2
S_2	5	3+3=6	3
S_3	5	6+4=10	4
S_4	5	10+5=15	5
S_5	5	15+6=21	6

1. *Estado inicial*:
```
n=int(input("Número: "))
termino=1
```

2. *Rango*: `range(2, n+2, 1)`
Como hay que dar `n` términos, tiene que haber `n` iteraciones.

Estructura repetitiva (bucle mientras)

3. *Cambio de estado*:

```
print(termino, end=" ")
termino=termino+cantidad
```

- Para cambiar el estado hay que cambiar cada una de las variables según se deduzca de la tabla, excepto la variable de control.

```
# establecer el estado inicial
n=int(input("Número: "))
termino=1
# range: range(2, n+2, 1)
for cantidad in range(2, n+2, 1):
    # cambiar el estado
    print(termino, end=" ")
    termino=termino+cantidad
print()
```

Ejemplos típicos de aplicación

Ejemplo

- Programa que calcula el factorial de un número leído por teclado.

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1 \quad (\text{si } n>0)$$
$$n! = 1 \quad (\text{si } n=0)$$

```
n= int(input("Introduzca un n° positivo o cero: "))
while (n<0):
   n= int(input("Introduzca un n° positivo o cero: "))

fact = 1
for i in range(1, n+1):
    fact = fact*i

print(n, "!= ", fact)
```

Ejemplos típicos de aplicación

Ejemplo

- Programa que calcula el *semifactorial* de un número leído por teclado.

$$n!! = n \cdot (n-2) \cdot (n-4) \cdot \ldots \cdot 1 \quad (\text{si } n>0)$$
$$n!! = 1 \quad (\text{si } n=0)$$

```
n= int(input("Introduzca un n° positivo o cero: "))
while (n<0):
   n= int(input("Introduzca un n° positivo o cero: "))

semifact = 1
for i in range(n, 0, -2):
    semifact = semifact*i

print(n, "!!= ", semifact)
```