UNIVERSITAT POLITÈCNICA DE VALÈNCIA

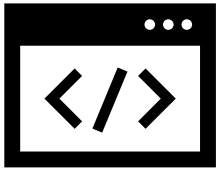
2021/2022

Programación

Tema 4. Estructuras de repetición (bucles, loops)

Esignificado de 2 simbolos

Ejercicio que hay que hacer en papel



Ejercicio que se puede hacer en ordenador

Instrucciones iterativas

- Los lenguajes de programación disponen de instrucciones de repetición:
  - estructuras iterativas, o
  - bucles (loops en ingles)
- En Python hay dos estructuras:
  - el bucle `while`
  - el bucle `for`
- Bucles permiten expresar de forma finita una ejecución tan larga como se requiera. El número de instrucciones a ejecutar puede estar determinado a priori o ser un número variable que depende de los datos

- Dos cuestiones a considerar:
  - El problema de la **terminación**: se debe garantizar que el número de pasadas sea finito.
  - La **eficiencia**: se debe ejecutar el menor número posible de instrucciones

La instrucción `while`

```
1  while condicion:
2      accion_1
3      accion_2
4      ...
5      accion_n
```

Ejecución:

- Se evalúa la `condición` y se repite la ejecución de las `acciones` hasta que se llega a un estado en que la `condición` es falsa.
- Al acabar, se continua con la instrucción que siguiente al `while`

mi-primer-bucle.py

```
1  # Ejemplo while de las
2  # transparencias del Tema 4
3
4  i = 1
5  while (i < 100):
6      i = i * 2
7      print (i)
8
```

Shell

```
>>> %Run mi-primer-bucle.py
  2
  4
  8
  16
  32
  64
  128

  fin
>>>
```

Se debe asegurar que en algún momento la condición deja de cumplirse para que la instrucción termine.

condición (i < 100)	i = i * 2	print(i)
1 < 100	2	2
2 < 100	4	4
4 < 100	8	8
8 < 100	16	16
16 < 100	32	32
32 < 100	64	64
64 < 100	128	128
~~128 < 100~~		
STOP		

mi-primer-bucle.py

```
1  # Ejemplo while de las
2  # transparencias del Tema 4
3
4  i = 1
5  while (i < 100):
6      i = i * 2
7      print (i)
8
```

Shell

```
>>> %Run mi-primer-bucle.py

  2
  4
  8
  16
  32
  64
  128

  fin

>>>
```

La instrucción `while` (ejercicio)

¿Qué ocurre con el siguiente programa?

```
i = 0
while (i < 10):
    print(i)
```

La instrucción `while` (ejercicio)

Haz una traza del siguiente programa para diferentes valores de i. ¿Qué ocurre si el valor de i es mayor o igual que 10?¿Y si es negativo?

```
i = int(input("Un valor inicial: "))

while (i < 10):
    print(i)
    i = i + 1
```

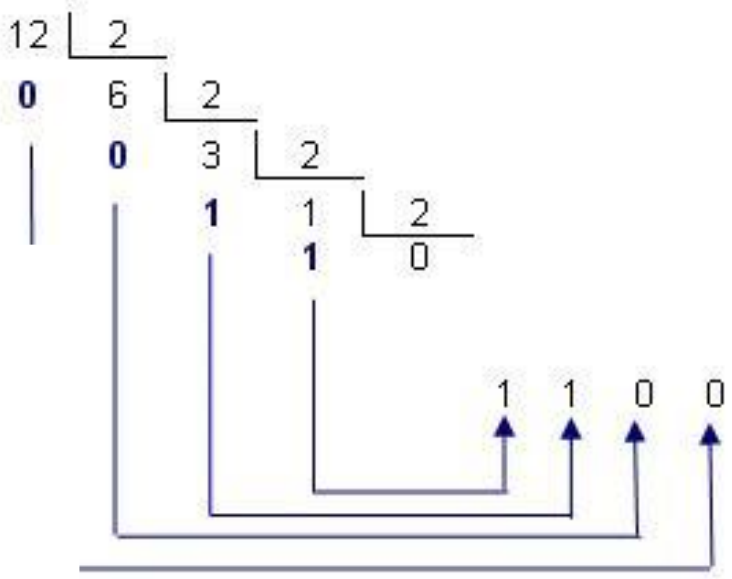
La instrucción `while` (ejercicio)

¿Qué hace el siguiente programa?

```
i = 100
while i > 10:
    print(i)
    i = i - 1
```

Ejemplo: La instrucción `while`.

- Convertir de decimal a binario
- Pensamos en una estrategia para la solucion



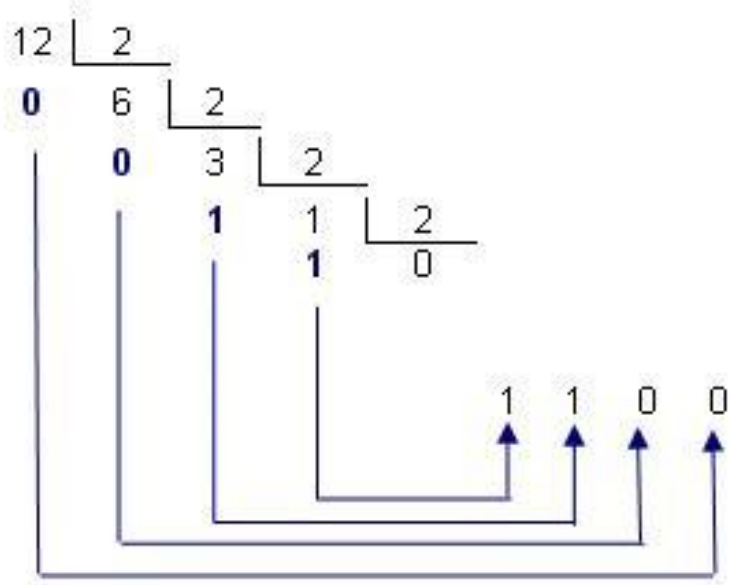
Dividor entre 2
hasta llegar a 0

o

Mientras (decimal // 2) no es 0,
Añadimos el resto %2 al binario

Ejemplo: La instrucción `while`.

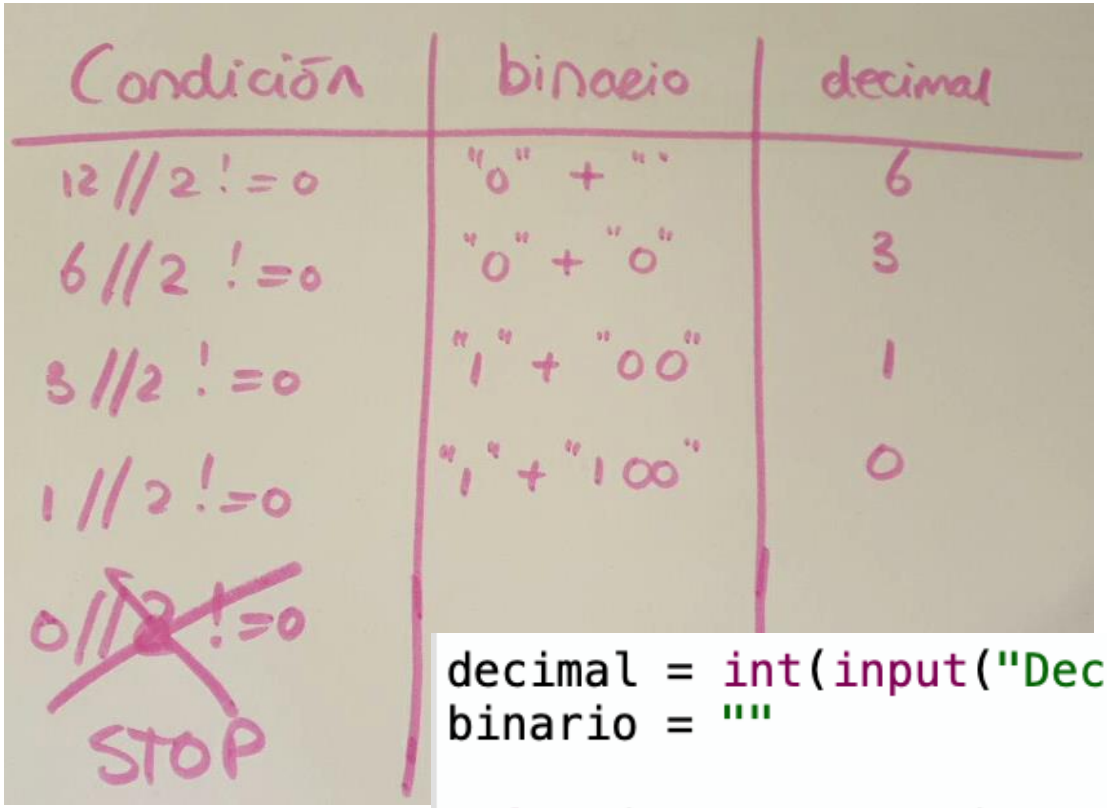
Convertir de decimal a binario

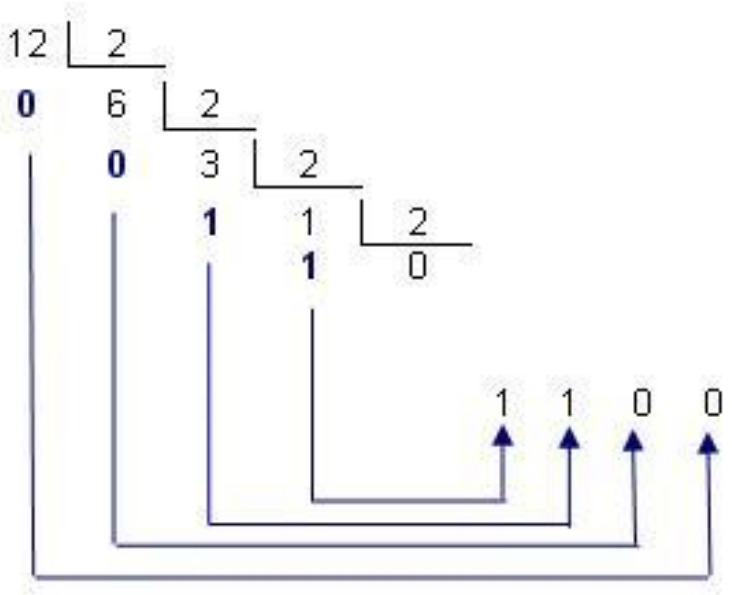



```
decimal = int(input("Decimal a convertir: "))
binario = ""

while (decimal != 0):
    binario = str(decimal % 2) + binario
    decimal = decimal // 2

print("en binario es {1}".format(decimal, binario))
```

Ejemplo: La instrucción `while`.

Convertir de decimal a binario






```
decimal = int(input("Decimal a convertir: "))
binario = ""

while (decimal != 0):
    binario = str(decimal % 2) + binario
    decimal = decimal // 2

print("en binario es {1}".format(decimal, binario))
```

La iteración como estrategia de diseño

- La **guarda del bucle**: condición que mientras evalúe a verdadero implicará la repetición de la ejecución del cuerpo del bucle (bloque de instrucciones).
- El **cuerpo del bucle**: las instrucciones que permiten avanzar paso a paso hacia la solución del problema.
- La **inicialización de variables**: antes de entrar en el bucle se debe asignar valores bien definidos a todas las variables involucradas en la ejecución del bucle.

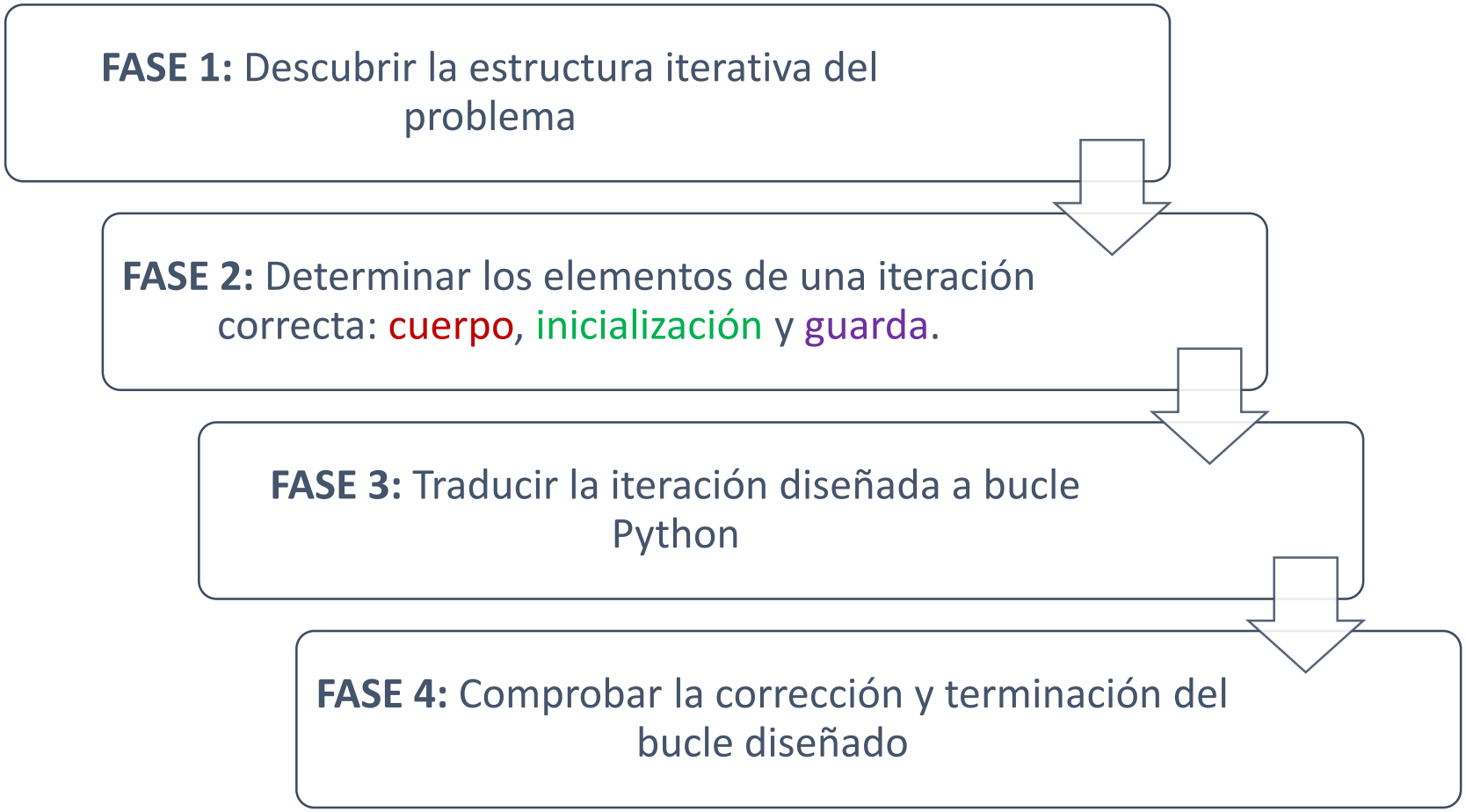
Inicialización de las variables del bucle

```
decimal = int(input("Decimal a convertir: "))
binario = ""

while (decimal != 0):
    binario = str(decimal % 2) + binario
    decimal = decimal // 2

print("en binario es {1}".format(decimal, binario))
```

Guarda del bucle

Cuerpo del bucle

La iteración como estrategia de diseño



La iteración como estrategia de diseño. Ejemplo. Fase 1: descubrir la estructura iterativa del problema

PROBLEMA: Multiplicar dos números enteros no negativos n y `m` **SIN** utilizar el producto matemático.

¡IDEA!

Multiplicar n por m es como sumar n veces m!

3 * 4 = 3 por 4 = sumar 3 veces 4 = 4 + 4 + 4

2 * 300 = 300 + 300

La iteración como estrategia de diseño. Ejemplo. Fase 2: determinar los elementos de la iteración

```
producto = 0

producto = producto + m
producto = producto + m
...
producto = producto + m
```

sumar n veces

Cuerpo, o qué instrucción(es) hay que repetir:

```
producto = producto + m;
```

Cuántas veces hay que repetir la ejecución del cuerpo: `n veces`

La iteración como estrategia de diseño. Ejemplo.
Fase 2: determinar los elementos de la iteración

Cuántas veces hay que repetir la ejecución del cuerpo: `n veces`

¿Cómo las cuento?

```
producto = 0
veces = 0

producto = producto + m
veces = veces+1
producto = producto + m

veces = veces+1
...
producto = producto + m

veces = veces+1
```

```
//he sumado 0 veces
//he repetido 0 veces

//he sumado 1 vez
//he repetido 1 vez
//he sumado 2 veces
//he repetido 2 veces

//he sumado n veces
//he repetido n veces
➔STOP
```

He sumado n veces

He repetido n veces

La iteración como estrategia de diseño. Ejemplo. Fase 2: determinar los elementos de la iteración

Refinando la fase 2:

Condición de parada ➔ Guarda

la ejecución del cuerpo *termina cuando* `(veces == n)`

el cuerpo *se ejecuta mientras* `(veces != n)`

Guarda

Cuerpo:
```
producto = producto + m
veces = veces + 1
```

Inicialización de las variables que intervienen en la iteración:

antes de empezar a repetir,
```
producto = 0
veces = 0
```

La iteración como estrategia de diseño. Ejemplo. Fase 3: Traducir a bucle Python

mult-con-suma.py

```
1  ############################################################
2  # Ejemplo de while en las transparencias del Tema 4
3  #
4  # Este programa multiplicar dos números enteros no negativos
5  # n y m SIN utilizar el producto matemático.
6  ############################################################
7
8
9  n = int(input("Introduce el primer valor: "))
10 m = int(input("Introduce el segundo valor: "))
11 producto = 0
12 veces = 0
13 while (veces != n):
14     producto = producto + m
15     veces += 1
16 print("El producto de %d y %d es igual a %d." %(n, m, producto))
```

//producto es una variable *acumuladora*

para acumular el resultado

//veces es una variable *contadora*

para contar la cantidad de sumas

La iteración como estrategia de diseño. Ejemplo. Fase 4: Testing, terminación y eficiencia

mult-con-suma.py

```
1  ############################################################
2  # Ejemplo de while en las transparencias del Tema 4
3  #
4  # Este programa multiplicar dos números enteros no negativos
5  # n y m SIN utilizar el producto matemático.
6  ############################################################
7
8
9  n = int(input("Introduce el primer valor: "))
10 m = int(input("Introduce el segundo valor: "))
11 producto = 0
12 veces = 0
13 while (veces != n):
14     producto = producto + m
15     veces += 1
16 print("El producto de %d y %d es igual a %d." %(n, m, producto))
```

- El código da un resultado correcto para $n \geq 0 \wedge m \geq 0$
- El bucle termina ya que $n \geq 0$ y veces comienza valiendo 0 y se incrementa en 1 en cada pasada del bucle: en a iteraciones dejará de cumplirse la guarda y el bucle terminará.
- La guarda del bucle se evalúa $n + 1$ veces y las instrucciones del cuerpo del bucle se ejecutarán n veces.

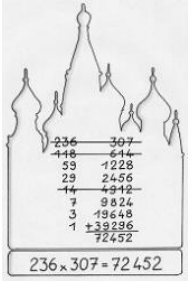
La instrucción **`while`**. ¿Recordamos?

Escribir un programa para multiplicar dos números utilizando la estrategia de multiplicación “a la russe” (*Actividad del Tema 1*)

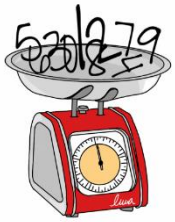
multiplicando	*multiplicador*	*producto*
981	1234 ✓	1234
490	~~2468~~	
245	4936 ✓	4936
122	~~9872~~	
61	19744 ✓	19744
30	~~39488~~	
15	78976 ✓	78976
7	157952 ✓	157952
3	315904 ✓	315904
1	631808 ✓	631808

1210554

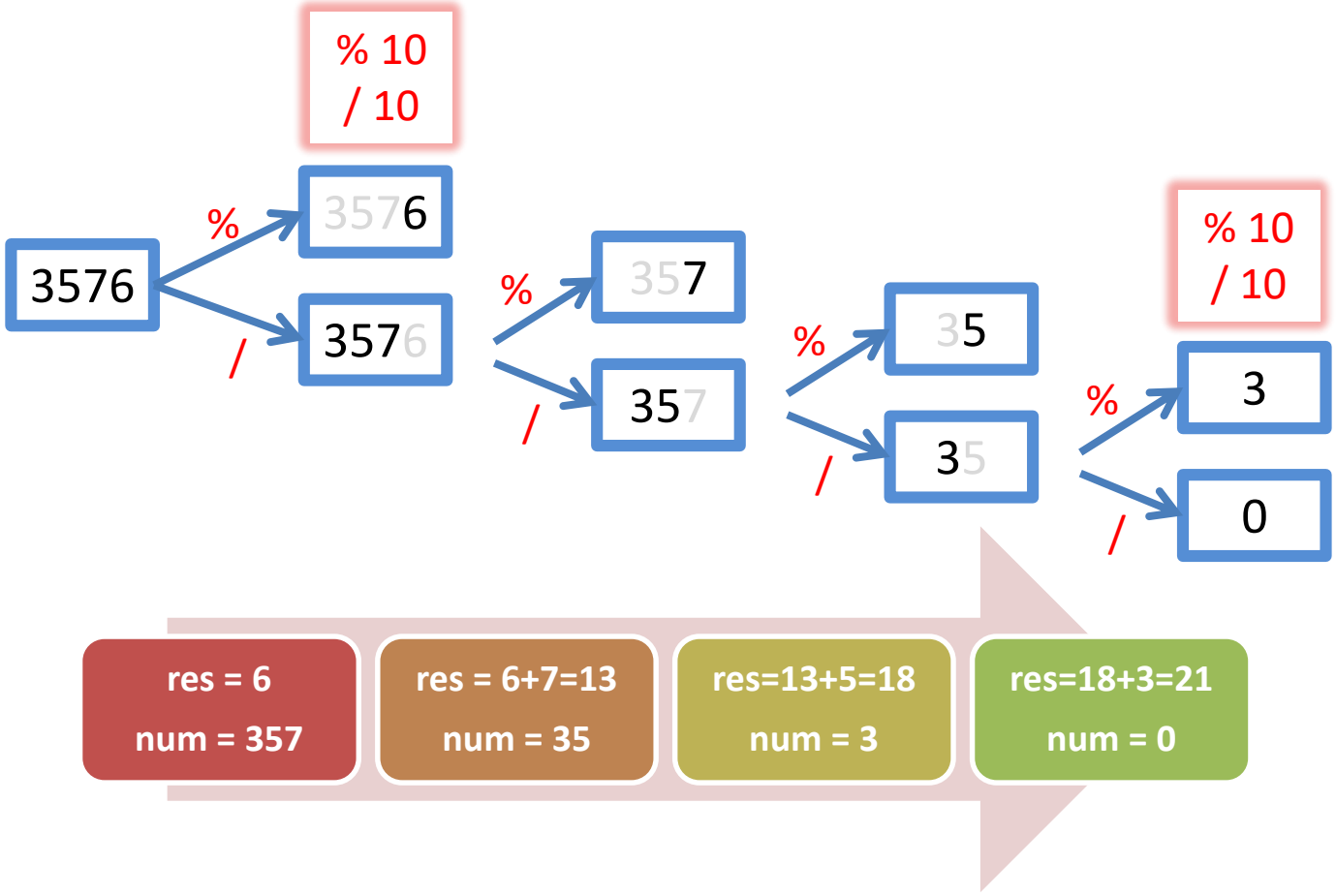
```
##########################################################
# Este programa aplica el algoritmo para multiplicacion
# a la Rusa.
#
# Entrada del programa:
#      el multiplicando y el multiplicador
# Salida del programa:
#      el producto del multiplicando y el multiplicador
# Restricciones:
#      no funciona cuando el multiplicando es negativo
#
# author: profesores de la asignatura de programacion
# date: 2019/2020
# version: 2
##########################################################

multiplicando = int(input("Cual es el multiplicando "))
multiplicador = int(input("Cual es el multiplicador "))
producto = 0

while multiplicando != 0:
    if multiplicando % 2 != 0:
        producto = producto + multiplicador
    multiplicando = multiplicando//2
    multiplicador = multiplicador * 2

print ("El resultado es ", str(producto))
```



La instrucción `while`. Ejemplo.



Escribir un programa para mostrar el número entero que resulta de sumar las dígitos de un decimal dado



Mientras num no es 0, suma num%10 a res y divide num por 10

La instrucción `while`. Ejemplo.

Escribir un programa para mostrar el número entero que resulta de sumar las cifras de uno dado

```
n = int(input("Introduce un numero entero: "))
resultado = 0
num = n
while num > 0:
    resultado += num % 10
    num = num // 10
print("La suma de las cifras de %d es %d." %(n, resultado))
```

- ¿Qué ocurriría si se introduce un número negativo?

La instrucción `while`. Ejemplo.

Modificamos el programa anterior para que el cálculo del número de cifras sólo se realice si el número introducido es positivo.

suma-digitos.py

```
1  ##################################################################
2  # Ejemplo en las transparencias del Tema 4
3  #
4  # Este programa pide al usuario un numero entero n, y devuleve
5  # el número entero que resulta de sumar las cifras de n
6  ##################################################################
7
8  n = int(input("Introduce un numero entero positivo: "))
9
10 while (n < 0):
11     print("Este programa solo funciona para numeros positivos!")
12     n = int(input("Introduce un numero entero positivo: "))
13
14 resultado = 0
15 num = n
16 while num > 0:
17     resultado += num % 10
18     num = num // 10
19 print("La suma de las cifras de %d es %d." %(n, resultado))
```

Shell

```
>>> %Run suma-digitos.py
  Introduce un numero entero positivo: 23456
  La suma de las cifras de 23456 es 20.
```

La instrucción `while` (ejercicio)

4. ¿Qué hacen los siguientes programas? Haz trazas.

```
k = 5
i = -4
while (i <= k):
    i = i + 2
    k = k - 1
    print (i+k)
```

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        break
    print(n)
print('Loop ended.')
```

```
i = 0
sum = 0
while i <= 4:
    sum += i
    i = i+1
print(sum)
```

```
que = input("¿Qué te parece?")

while (que != "me encanta"):
    print("¡¡Venga, no seas negativ@!!")
    que = input("Cuentame, ¿Qué te parece?")

print("¡¡Siii!! ¡A mi tambien", que + "!")
```

La instrucción `for-in`

```
1  for variable in serie de valores:
2      accion_1
3      accion_2
4      ...
5      accion_n
```

verduras.py

```
1  # Ejemplo en las transparencias del Tema 4
2
3  print("Verduras con mucha vitamina C son:\n")
4  for nombre in ["Pimiento", "Brocoli", "Espinaca", "Tomates"]:
5      print(nombre)
```

Consola

```
>>> %Run verduras.py
  Verduras con mucha vitamina C son:

  Pimiento
  Brocoli
  Espinaca
  Tomates
```

Ejecución:

- En cada iteración, la variable va tomando como valor los de la serie de forma ordenada de izquierda a derecha.
- Se ejecutan las `acciones` para todos los valores de la serie.
- Al acabar, se continua en la instrucción siguiente a la iterativa.

La instrucción `for-in`. La función `range`

La función `range` permite generar una serie de valores.

- `range(inicio, fin)`:
  - devuelve la serie de enteros desde `inicio` hasta `fin-1`.

- `range(fin)`:
  - devuelve la serie de enteros desde `0` hasta `fin-1`.

```
>>> for i in range (3):
        print (i)


0
1
2

>>>
```

```
>>> for i in range (1,3):
        print (i)


1
2

>>>
```

```
>>> for i in range (-3,3):
        print (i)


-3
-2
-1
0
1
2

>>>
```

La instrucción `for-in`. La función `range`

La función `range` permite generar una serie de valores.

- `range(inicio, fin, inc)`:
  - devuelve la serie de enteros desde `inicio` hasta el anterior a `fin`, tomando como incremento `inc`.

```
>>> for i in range (1,5,1):
        print(i)


1
2
3
4
```

```
>>> for i in range (6, 1, -1):
        print (i)


6
5
4
3
2
>>>
```

```
>>> for i in range (1,5,2):
        print(i)


1
3
```

```
>>> for i in range (6, 1, -2):
        print (i)


6
4
2
>>>
```

```
>>> for i in range (1, 6, -1):
        print (i)

>>>
```

La instrucción `for-in. Range con string`

for-string.py

```
1  ##################################################
2  # Ejemplo de for en las transparencias del Tema 4
3  ##################################################
4
5  s = input("Type a string: ")
6  for i in range(0, len(s)):
7      if (s[i].isdigit()):
8          print(s[i])
```

¿Que hace este programa?

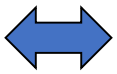
La instrucción `for-in`. Equivalencia con `while`

```
i = valor_inicial
while i <= valor_final:
    acciones
    i = i + 1
```

```
for i in range(valor_inicial, valor_final+1):
    acciones
```

sumatorio-while.py

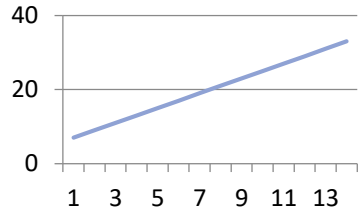
```
1  # ejemplo del Tema 4
2
3  sumatorio = 0
4  i = 1
5  while i <= 1000:
6      sumatorio = sumatorio + i
7      i = i + 1
8  print(sumatorio)
```



sumatorio-for.py

```
1  # ejemplo del Tema 4
2
3  sumatorio = 0
4  for i in range(1, 1001):
5      sumatorio = sumatorio + i
6  print(sumatorio)
```

La instrucción `for-in`. Ejemplo.



Escribir un programa para calcular el término n-ésimo de la serie:

$$a_1 = 7, \quad a_i = a_{i-1} + i \text{ , } i \geq 2$$

$n = 6$

$a_1 = 7$

$a_2 = a_1 + 2 = 7 + 2 = 9$

$a_3 = a_2 + 3 = 9 + 3 = 12$

$a_4 = a_3 + 4 = 12 + 4 = 16$

$a_5 = a_4 + 5 = 16 + 5 = 21$

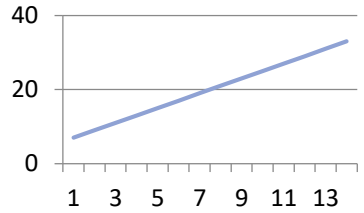
$a_6 = a_5 + 6 = 21 + 6 = 27$

iteración	i	term
0	1	7
1	2	9
2	3	12
3	4	16
4	5	21
5	6	27

`i<n`

`i==n`

La instrucción `for-in`. Ejemplo.



Escribir un programa para calcular el término n-ésimo de la serie:

$$a_1 = 7, \quad a_i = a_{i-1} + i \ , \ i \geq 2$$

n-esimo-serie.py

```
1  ##################################################
2  # Ejemplo en las transparencias del Tema 4
3  #
4  # Este programa calcula el término n-ésimo de
5  # la serie:
6  #  - a_1 = 7
7  #  - a_i = a_i+1 + i (para i>=2)
8  ##################################################
9
10 n = int(input("Introduce un entero: "))
11 term = 7 # valor del ultimo termino calculado. term = a_1 = 7
12 i = 1 #indice del ultimo termino term = a_1
13
14 for i in range (2, n+1):
15     term = term + i
16
17 print ("El termino %d de la seria es %d." % (n, term))
```

break, continue

```
1 for i in range(7,14):
2     if i % 3 == 0:
3         break
4     print(i)
```

```
>>> %Run break.py
7
8
```

Cuando se ejecuta **break** abandona el bucle

```
1 for i in range(7,14):
2     if i % 3 == 0:
3         continue
4     print(i)
```

```
>>> %Run continue.py
7
8
10
11
13
```

Cuando se ejecuta **continue** abandona la iteración del bucle y pasa a la siguiente

La instrucción `while-else`

```
while condicion:
    accion_1
    accion_2
    ....
    accion_n
else:
    otra_accion

accion_despues
```

Ejecución:

- El bloque de código del **`else`** se ejecutará **después de que el bucle haya finalizado** y *solo si* la condición del bucle es falsa (lo que implica que no se ha salido del **`while`** mediante un break).

La instrucción `for-else`

```
for variable in serie_de_valores:
    accion_1
    accion_2
    ....
    accion_n
else:
    otra_accion

accion_despues
```

Ejecución:

El código del **else** se ejecutará **después de que el bucle haya finalizado** y *solo si* no quedan valores en la sucesión del in, lo que implica que no se ha salido del for con un break.

La instrucción `for-in`. Ejemplo.

$$\sum_{i=1}^{n} a_i$$

Escribir un programa para calcular la suma de los n primeros términos de la serie:

$$a_1 = 7, \quad a_i = a_{i-1} + i \text{ , } i \geq 2$$

n = 6
a_1 = 7, suma = 7
$a_2 = a_1+2$ = 9, suma = 7+9 = 16
$a_3 = a_2+3$ = 12, suma = 16+12 = 28
$a_4 = a_3+4$ = 16, suma = 28+16 = 44
$a_5 = a_4+5$ = 21, suma = 44+21 = 65
$a_6 = a_5+6$ = 27, suma = 65+27 = 92

iteración	i	term	suma
0	1	7	7
1	2	9	16
2	3	12	28
3	4	16	44
4	5	21	65
5	6	27	92

La instrucción `for-in`. Ejemplo.

suma-n-esimo-serie.py

```
1  ###################################################
2  # Ejemplo en las transparencias del Tema 4
3  #
4  # Este programa calcula la suma de los n primeros
5  # términos de la serie:
6  #  - a_1 = 7
7  #  - a_i = a_i+1 + i (para i>=2)
8  ###################################################
9
10
11 n = int(input("Introduce un entero: "))
12
13 #asegurar que el usuario nos da un valor n>=1
14 while (n <= 0):
15     print("Solo valores de mayor a 0!")
16     n = int(input("Introduce un entero: "))
17
18
19 #definimos a_i con i y term
20 term = 7 # valor del ultimo termino calculado. term = a_1 = 7
21 i = 1 #indice del ultimo termino term = a_1
22
23 #inicializamos la variable acumuladora: suma
24 suma = term
25
26 for i in range (2, n+1):
27     term = term + i
28     suma = suma + term
29
30 print("La suma de los %d primeros terminos de la serie es %d" % (n, suma))
```

$$\sum_{i=1}^{n} a_i$$

La instrucción `for` (ejercicio)

¿Qué hace el siguiente programa? Haz trazas.

a. ¿Cuál será el resultado del programa si los datos introducidos son 3 y 6? ¿Por qué?
b. ¿Y si los datos introducidos fuesen 7 y 7?
c. ¿El resultado del programa depende del orden en que son introducidos los datos? ¿Por qué?
d. Expresar con una fórmula qué cálculo hace este programa cuando a <= b.

```
a=int(input("Introduce un número entero:"))
b=int(input("Introduce otro número entero:"))
if a > b:
    aux=a
    a=b
    b=aux
resultado=1
for i in range(a+1,b+1):
    resultado=resultado*i
print("El resultado es:",resultado)
```

Anidamiento de bucles. Ejemplo

- Cualquiera de las instrucciones que forman el cuerpo de un bucle puede ser, a su vez, una instrucción iterativa o de repetición.

- Ejemplo: mostrar por pantalla las tablas de multiplicar desde 1 al 10.

```
>>> %Run tablas-multiplicar.py
La tabla de  1
 1 x  1 =  1
 1 x  2 =  2
 1 x  3 =  3
 1 x  4 =  4
 1 x  5 =  5
 1 x  6 =  6
 1 x  7 =  7
 1 x  8 =  8
 1 x  9 =  9
 1 x 10 = 10
La tabla de  2
 2 x  1 =  2
 2 x  2 =  4
 2 x  3 =  6
 2 x  4 =  8
 2 x  5 = 10
 2 x  6 = 12
 2 x  7 = 14
 2 x  8 = 16
 2 x  9 = 18
 2 x 10 = 20
La tabla de  3
 3 x  1 =  3
 3 x  2 =  6
 3 x  3 =  9
 3 x  4 = 12
 3 x  5 = 15
 3 x  6 = 18
 3 x  7 = 21
 3 x  8 = 24
 3 x  9 = 27
 3 x 10 = 30
```

tablas-multiplicar.py

```
1  ##################################################
2  # ejemplo de las transparencia del tema 4
3  #
4  # Este programa muestra por pantalla las tablas de
5  # multiplicar desde 1 al 10.
6  ##################################################
7
8  for i in range (1, 11):
9      print ("La tabla de %2d" % i)
10     for j in range (1, 11):
11         print ("%2d x %2d = %2d" % (i, j, i*j))
```

Anidamiento de bucles. Ejercicio

Haz una traza de los segmentos de código siguientes. ¿Qué muestran por pantalla?

```
nfil = 4
ncol = 3
for i in range (1, nfil + 1):
    for j in range (1, ncol + 1):
        print("%d - %d = %d" % (i, j, i - j))
```

```
n = int(input("Introduce un entero: "))

for i in range(n+1):
    for j in range (0, i + 1):
        print('K')
    for j in range (0, n + 1):
        print('U')
    for j in range (0, n - i + 1):
        print('T')
```

La instrucción `for` (ejercicio)

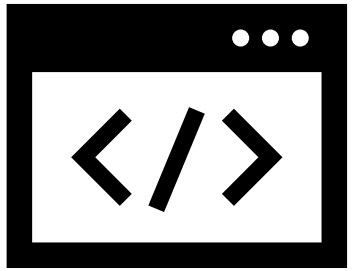
¿Qué hace el siguiente programa? Haz trazas.

```
n=int(input("Introduce un numero entero:"))
a=n
i=0
while a//10!=0:
    a=a//10
    i=i+1
b=n
m=0
print ("i is ", i)
for j in range(i,-1,-1):
    print("j is ", j)
    d=1
    for k in range(0,j):
        d=d*10
    m=m+(b%10)*d
    b=b//10
print ("El resultado es: {0}".format(m))
```

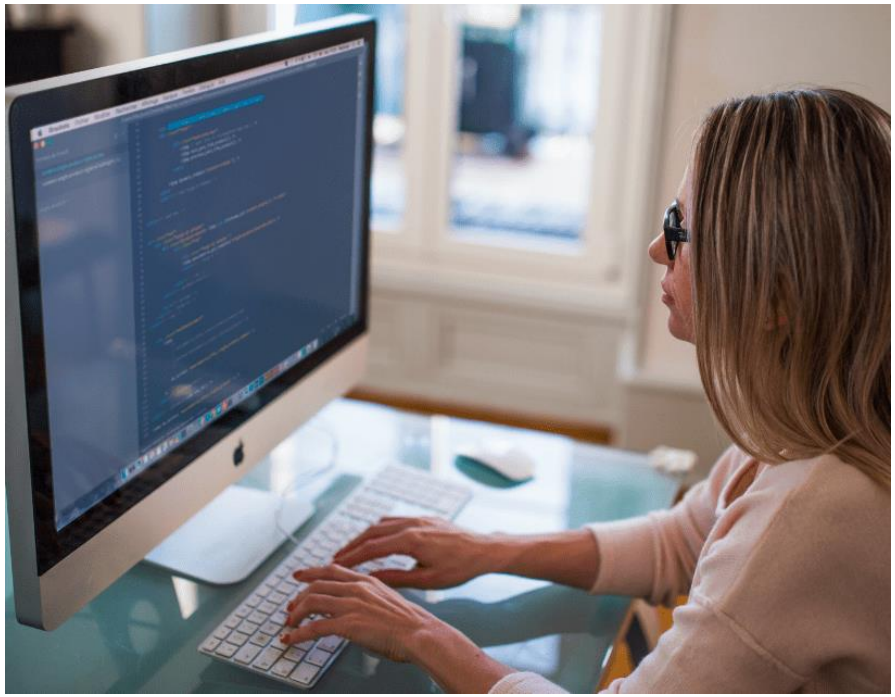
Ejercicio – Tablas de pitagoras

- Escribe un programa que imprime la tabla de multiplicar, como la **tabla pitagórica** (denominada así en honor de Pitágoras). La primera fila y la primera columna contienen los números que se van a multiplicar (habitualmente, los números de 1 hasta el 10), y en la intersección de cada fila y cada columna está el producto del número de su fila por el número de su columna.

```
>>> %Run tabla-multiplicacion-pitagorica.py
  1       2       3       4       5       6       7       8       9       10
  2       4       6       8       10      12      14      16      18      20
  3       6       9       12      15      18      21      24      27      30
  4       8       12      16      20      24      28      32      36      40
  5       10      15      20      25      30      35      40      45      50
  6       12      18      24      30      36      42      48      54      60
  7       14      21      28      35      42      49      56      63      70
  8       16      24      32      40      48      56      64      72      80
  9       18      27      36      45      54      63      72      81      90
  10      20      30      40      50      60      70      80      90      100
```



¡Solo hay una forma de aprender a programar!



practicando, practicando, practicando, practicando