3 Informática (Teoría)

Python: Fors

Joaquim Gabarro
gabarro@cs.upc.edu

Computer Science
Universitat Politècnica de Catalunya

Ejemplos con `for`

Example: `drawH(n)`

Write a function `drawH(n)` that given an odd integer `n`$\geq$ `3` returns a string representing a letter `H` of size `n` formed with symbol `*`. Follow the pattern of the example below.

```
>>> print(drawH(5))
*   *
*   *
*****
*   *
*   *
>
```

El salto de línea y el print

El siguiente carácter corresponde al salto de línea:

```
"\n"
```

Para que este carácter se interprete como un salto de linea es necesario usar `print`.

Ejemplo:

```
>>> n=5
>>> w= 5*"*"
>>> w
'*****'
>>> w= w + "\n" + w
>>> w
'*****\n*****'
>>> print(w)
*****
*****
```

Un Primer Intento con `for`

Introducimos una variable `mid = n//2` (`mid` denota `middle`) y una variable `line` para recorrer el `for` tenemos:
Finalmente (fichero `drawHbad.py`):

```
def drawH(n):
    mid = n//2
    sH=""
    for line in range(mid):
        sH= sH + ('*' + (n - 2)*' ' + '*'+ '\n')
    sH=sH + (n*'*' + '\n')
    for line in range(mid):
        sH= sH + ('*' + (n - 2)*' ' + '*'+ '\n')
    return sH
```

Problema:

Al final hay un salto de línea de más

```
>>>
 RESTART: C:...\bad-drawH.py
>>> print(drawH(7))
*     *
*     *
*     *
*******
*     *
*     *
*     *

>>>
```

Solución

La última límea no tiene salto de línia. Como `n` ≥ 3 tratamos la última linea de modo distinto (fichero `drawH(n).py`).

```
def drawH(n):
    mid = n//2
    sH=""
    for line in range(mid):
        sH= sH + ('*' + (n - 2)*' ' + '*'+ '\n')
    sH=sH + (n*'*' + '\n')
    for line in range(mid-1):
        sH= sH + ('*' + (n - 2)*' ' + '*'+ '\n')
    sH= sH + ('*' + (n - 2)*' ' + '*')
    return sH
```

Ejemplo:

```
>>>
 RESTART: C:...\drawH.py
>>> print(drawH(7))
*     *
*     *
*     *
*******
*     *
*     *
*     *
>>>
```

Sin `for` (explícito)

```
def drawH(n):
    mid = n//2
    sH = ""
    sH = sH + mid * ('*' + (n - 2)*' ' + '*'+ '\n')
    sH = sH + (n*'*' + '\n')
    sH = sH + (mid-1) * ('*' + (n - 2)*' ' + '*'+ '\n')
    sH = sH + ('*' + (n - 2)*' ' + '*')
    return sH
```

Class Exercise: draw_square

Design a function draw_square(n) that given $n \geq 1$ returns a string such that:

```
>>> print(draw_square(1))
*
>>> print(draw_square(3))
***
***
***
>>> print(draw_square(5))
*****
*****
*****
*****
*****
>>>
```

Class exercise: `draw_triangle`

Design a function `draw_triangle(n)` that given $n \geq 1$ returns a string such that:

```
>>> print(draw_triangle(1))
*
>>> print(draw_triangle(2))
 *
***
>>> print(draw_triangle(5))
    *
   ***
  *****
 *******
*********
>>>
```

Ejemplo: Contando números primos

Dado `n>1`, se pide diseñar una función `count_primes(n)` que retorne cuantos números primos hay entre 2 y `n`.
Por ejemplo `n=10` :

$$2, 3, 4, 5, 6, 7, 8, 9, 10$$

Entre 2 y 10 los primos son 2, 3, 5, 7 por tanto,
`count_primes(10)` ha de retornar 4

Recordemos, un número natural n es primo, si es mayor que 1 y sus únicos divisores son 1 y n.

Parte I: recorrido

Cuando `n=10` contamos cuantos primos hay entre el 2 y el 10.

Recordemos

```
>>> list(range(2,n+1))
[2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>
```

Suponiendo que tenemos una función booleana `is_prime(n)`, hacemos un recorrido:

```
def count_primes (n):
    count=0
    for m in range (2,n+1):
        if is_prime(m): count=count+1
    return count
```

Parte II: búsqueda

Diseñemos la función `is_prime(n)` con `n=9`.
Hay que buscar el primer número `k`, tal que `1 < k < n` de la lista `[2, 3, 4, 5, 6, 7, 8]` que divida a `n`

```
>>> n=9
>>> list(range(2,n))
[2, 3, 4, 5, 6, 7, 8]
>>>
```

Atención: Hay que quitar `n=9` de la lista (`n=9` siempre es divisible por `n=9`). El rango es `range(2,n)` y no `range(2,n+1)`.

```
def is_prime(n):
    if n <= 1: return false
    for k in range(2, n):
        if n % k == 0: return False
    return True
```

Restrinjamos (siempre que sea posible) el rango de la búsqueda.

Miremos la descomposición de `range(2,n)` en range(2, n//2+1) y range(n//2+1, n)

Caso par: `n=10` tenemos `[2, 3, 4, 5 | 6, 7, 8, 9]`

```
>>> n=10
>>> list(range(2, n//2+1))
[2, 3, 4, 5]
>>> list(range(n//2+1, n))
[6, 7, 8, 9]
```

Hace falta buscar `k in [6, 7, 8, 9]`? Par que `n=10` sea divisible por 6 hace falta encontrar un `z` $\geq$ 2 tal que `6*z=` 10 pero como `6*2 =` 12 esto es imposible. Los casos 7, 8, 9 son similares.

Caso impar: `n=9` tenemos `[2, 3, 4 | 5, 6, 7, 8]`

```
>>> n=9
>>> list(range(2, n//2+1))
[2, 3, 4]
>>> list(range(n//2+1, n))
[5, 6, 7, 8]
```

Hace falta buscar `k in [5, 6, 7, 8]`? Par que `n=9` sea divisible por 5 hace falta encontrar un `z` $\geq$ 2 tal que `5*z=` 9 pero como `5*2 =` 10 esto es imposible. Los casos 6, 7, 8 son similares.

Los ejemplos precedentes se pueden generalizar y formalizar sin problema.

Conclusión: Independientemente de la paridad, es suficiente recorrer range(2, n//2+1) con lo que tenemos:

```
def is_prime(n):
    if n <= 1: return false
    for k in range(2, n//2+1):
        if n % k == 0: return False
    return True
```

Finalmente (fichero count_primes.py):

```
def is_prime(n):
    if n <= 1: return false
    for k in range(2, n//2+1):
        if n % k == 0: return False
    return True

def count_primes (n):
    count=0
    for m in range (2,n+1):
        if is_prime(m): count=count+1
    return count
```

```
>>>
= RESTART: C:/....count_primes.py =
>>> count_primes(10)
4
>>> count_primes(100)
25
>>> count_primes(1000)
168
>>> count_primes(10000)
1229
>>> count_primes(100000)
9592
>>>
```

Primeros pasos en integración mumérica

Regla del rectángulo

El intervalo entre $x = a$ y $x = b$ se divide en n partes iguales por los puntos $a = x_0, x_1, \ldots, x_{n-1}, x_n = b$

$$\int_a^b f(x)\, dx \approx \sum_{i=0}^{i=n} \frac{b-a}{n} f(x_i)$$

- Notad $\frac{b-a}{n} f(x_i)$ corresponde al area de un rectángulo de base $\frac{b-a}{n}$ y altura $f(x_i)$

$$area_i = \frac{b-a}{n} f(x_i) = base \times altura_i$$

- Notad que $x_i = x_0 + i \times \frac{b-a}{n} = a + i \times \frac{b-a}{n}$
- En particular, $x_n = a + n \times \frac{b-a}{n} = b$

$$\int_a^b f(x)\ dx \approx area_0 + area_1 + \cdots + area_n$$

El programa consiste en un recorrido $[0, \ldots, n]$ que va calculando las $area_i$, $0 \leq i \leq n$ i acumula en una variable area_total los valores

$$\begin{aligned}
area_total &= 0 \\
area_total &= area_0 \\
area_total &= area_0 + area_1 \\
&\vdots \\
area_total &= area_0 + area_1 + \cdots + area_n
\end{aligned}$$

Caso: $f(x) = x^2$ (square function)

Suponganos $n = 10$, hay que evaluar

$$f(x_0), \ldots, f(x_i), \ldots, f(x_{10})$$

Recordad

```
>>> n= 10
>>> list(range(0,n+1))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
def integral_definida_sq(a,b,n):
    x = a
    base = (b - a) / n
    area_total = 0
    for i in range(0, n+1):
        #calcular area rectangulo i-esimo
        area_total = area_total +  # area rectangulo i-esimo
        x = x + base
    return area
```

Fichero integral_definida_sq.py

```
def integral_definida_sq(a,b,n):
    x = a
    base = (b - a) / n
    area_total = 0
    for i in range(0, n+1):
        altura = x * x
        area = base * altura
        area_total = area_total + area
        x = x + base
    return area_total
if __name__ == "__main__":
    print(integral_definida_sq(0,1,1))
    print(integral_definida_sq(0,1,10))
    print(integral_definida_sq(0,1,100))
    print(integral_definida_sq(0,1,1000))
    print(integral_definida_sq(0,1,10000))
    print(integral_definida_sq(0,1,100000))
    print(integral_definida_sq(0,1,1000000))
```

El resultado es:

```
>>>
 RESTART: C:...\integral_definida_sq.py
1.0
0.3849999999999999
0.3383500000000004
0.33383350000000095
0.3333833349999431
0.3333383333493713
0.3333338333339987
>>>
```

Recordad $\int_a^b x^2 \, dx = (b^3 - a^3)/3$ entonces $\int_0^1 x^2 \, dx = 1/3$

```
>>> 1/3
0.3333333333333333
>>>
```

Para apasionados: lambda abstractions

Recapitulemos.

- Hemos visto la función `integral_definida_sq(a,b,n)`, que era útil para integrar la función $f(x) = x^2$.
- Si queremos cambiar de función de $f(x) = x^2$ a $f(x) = x^3$ siempre podemos crear una cópia de `integral_definida_sq(a,b,n)`, bautizarla com `integral_definida_cube(a,b,n)` y modificar

  ```
  altura = x * x
  ```

  por

  ```
  altura =  x*x*x
  ```

- Pregunta. No se puede pasar la función f como parámetro? Es decir, queremos una función `integral_definida(a,b,n,f)` en que `f` sea un parámetro

Lambda abstractions

```
def my_square():
    return lambda x: x*x

=== RESTART: C:/..../my_square.py ===
>>> f=my_square()
>>> f(2)
4
>>> f(9)
81
```

En el fichero `integral_definida.py` tenemos la función `integral_definida(a,b,n,f)` que vemos seguidamente:

```
def my_square():
    return lambda x: x*x

def my_cube():
    return lambda x: x*x*x

def integral_definida(a,b,n,f):
    x = a
    base = (b - a) / n
    area_total = 0
    for i in range(0, n+1):
        altura = f(x)
        area = base * altura
        area_total = area_total + area
        x = x + base
    return area_total

if __name__ == "__main__":
    print(integral_definida(1, 100, 10000, my_square()))
    print(integral_definida(1, 2, 1000, my_cube()))
```