Introducción a la Programación

Práctica 3 – Ciclos

Versión del 19 de julio de 2016

1. Ciclos

Las computadoras son de gran utilidad para la automatización de tareas repetitivas. Repetir tareas idénticas o similares sin cometer errores es algo que las computadoras hacen muy bien y las personas no.

Cada repetición se llama **iteración**, y Python provee algunas herramientas del lenguaje que simplifican la escritura de programas iterativos.

Las dos herramientas que vamos a ver son las sentencias `while`(mientras) y `for`(para).

1.1. La sentencia `while` (mientras)

Usando una sentencia `while`, podemos escribir un programa que haga una cuenta regresiva:

```
n = int(input("Ingrese un valor para la cuenta regresiva"))
while n > 0:
  print(n)
  n = n-1
print("¡Despegue!")
```

Una sentencia `while` se puede leer casi como si estuviese en castellano[1]. Lo que significa, "Mientras `n` es mayor que cero, continuar imprimiendo el valor de `n` y reduciendo el valor de `n` en 1. Cuando llegue a cero, imprimir la palabra '¡Despegue!' ". Aquí hay un detalle muy importante = significa asignar y muchas veces se confunde con el = de matemática, $n = n - 1$ en matemática sería absurdo pero en programación significa que al valor que tiene actualmente n se le reste uno y se asigne a n de esta manera en la próxima iteración n vale 1 menos.

Más formalmente, el flujo de ejecución para una sentencia `while` es el siguiente:

1. Evaluar la condición entre paréntesis produciendo `True` o `False`.
2. Si la condición es falsa, salir de la sentencia `while` y continuar la ejecución en la siguiente sentencia.
3. Si la condición es verdadera, ejecutar cada una de las sentencias que están tabuladas a continuación.

A este tipo de flujos se los llama **ciclos** porque el tercer paso arma un ciclo al volver al paso 1. Notá que si la condición es falsa la primera vez, las sentencias dentro del ciclo no se ejecutan nunca. Las sentencias dentro del ciclo son usualmente llamadas el **cuerpo** del ciclo.

El cuerpo del ciclo debe cambiar el valor de una o más variables para que, eventualmente, la condición se haga falsa y el ciclo termine. . Si no, el ciclo se repetirá por siempre, lo cual es conocido como un ciclo **infinito**. A esta variable se la llama *Variable de Control*

En el caso de nuestra cuenta regresiva, podemos probar que el ciclo terminará ya que sabemos que el valor de `n` es finito, y podemos ver que el valor de `n` toma valores cada vez más pequeños cada vez que pasa por el ciclo (en cada **iteración**), con lo cual eventualmente llegaremos a cero. En otros casos no es tan sencillo de ver:

[1] N.d.T.: la palabra *while* significa *mientras* en castellano.

```
n = int(input("Ingrese un numero mayor a cero"))
while n != 1:
  print n,
  if n % 2 == 0:        # n es par
    n = n / 2
  else:                 # n es impar
    n = n * 3 + 1
```

La condición de este ciclo es `n != 1`, así que el ciclo continuará hasta que `n` sea 1, lo que hará falsa la condición.

En cada iteración, el programa imprime el valor de `n` y luego verifica si es par o impar. Si es par, el valor de `n` se divide por dos. Si es impar, el valor se reemplaza por $3n + 1$. Por ejemplo, si el valor inicial es 3, la secuencia resultante es 3, 10, 5, 16, 8, 4, 2, 1.

Como `n` a veces aumenta y a veces disminuye, no hay una demostración obvia de que `n` alcance alguna vez 1, o de que el programa vaya a terminar. Para algunos valores particulares de `n`, podemos probar que termina. Por ejemplo, si el valor inicial es una potencia de dos, entonces el valor de `n` será par cada vez que pase por el ciclo hasta llegar a valer 1. El ejemplo anterior termina con una secuencia que empieza en 16.

Dejando de lado los valores particulares, la pregunta interesante es si podemos demostrar que este programa termina para *todo* valor de n. Hasta ahora, nadie fue capaz de demostrarlo ¡o de refutarlo!

1.2. La función `range()`

Esta función de Python nos da un rango de valores y es ideal para iterar con la sentencia `for` que veremos a continuación.

Se puede utilizar con diferente cantidad de parámetros, en su versión completa, toma 3 parámetros, los cuales deben ser siempre enteros, y tiene la siguiente estructura:

```
range(INICIO, FIN, PASO)
```

representa un rango entre INICIO hasta FIN (sin incluirlo), con saltos de tamaño PASO.

Si se omite el parámetro PASO, su valor por defecto es 1. Si se omiten los parámetros INICIO y PASO, range asume que INICIO es 0 y PASO es 1.

Por ejemplo:

```
range(0, 10, 1)
```

representa el rango que incluye a los valores: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9.

```
range(4, 12, 2)
```

representa el rango que incluye a los valores: 4, 6, 8 y 10.

```
range(5)
```

representa el rango que incluye a los valores: 0, 1, 2, 3 y 4.

```
range(-2, 2)
```

representa el rango que incluye a los valores: -2, -1, 0 y 1.

En caso de que usemos un PASO negativo, el rango será decreciente, es decir, de mayor a menor.

Ejemplo:

```
range(10, 1, -1)
```

representa el rango que incluye a los valores: 10, 9, 8, 7, 6, 5, 4, 3 y 2.

1.3. Ciclos `for` (para)

Los ciclos que hemos escrito hasta ahora tienen una cierta cantidad de elementos en común. Todos ellos comienzan inicializando una variable; tienen una condición que depende de esa variable y dentro del ciclo hacen algo con esa variable, como incrementarla.

Otra forma de escribir ciclos de forma más concisa es utilizando la sentencia `for` (para).

```
for VARIABLE in CONJUNTO:
  CUERPO
```

Por ejemplo, si quisieramos hacer un programa que cuente desde 1 hasta 100 con un `for`:

```
for i in range(1,101):
  print(i)
```

esto se lee: "Para i desde 1 hasta 101, mostrar i."

Aquí usamos a la función `range()` para obtener un conjunto de valores.

En este caso, la variable `i` se la denomina *variable de control* y no es necesario que le asignemos los valores del rango. Estos valores se le van asignando automáticamente en cada iteración.

Como cualquier otra sentencia `while`, la cuenta regresiva que hicimos antes se puede escribir con una sentencia `for`. Esta se vería de la siguiente manera:

```
n = int(input("Ingrese un numero mayor a cero"))
for i in range(n,0,-1):
  print(i)
print("¡Despegue!")
```

Este programa produce el mismo resultado que su versión con la sentencia `while`. Sin embargo, veamos que en este caso no es necesario re-asignar la variable `i` en cada iteración.

1.4. Acumuladores

Los acumuladores son variables que suelen utilizarse en los ciclos para ir guardando el resultado parcial de algún cálculo.

Por ejemplo, si quisieramos hacer un programa que calcule la suma de los primeros `n` números naturales, podríamos hacerlo de la siguiente manera con una sentencia `for`:

```
n = int(input("Ingrese un numero natural"))
suma = 0;
for i in range(1,n):
  suma = suma + i
print("La suma de los primeros", n ,"numeros naturales es", suma)
```

En este caso, la variable `suma` es inicializada en 0 y en cada iteración se le va asignando su valor calculado hasta el momento más el valor de la variable `i`. Debería quedar claro que la variable `suma` acumula el resultado de la suma realizada hasta el momento y por eso lleva el nombre de *acumulador*

2. Cadenas

Hasta ahora hemos dicho que las cadenas sirven para guardar texto en variables. Cualquier cosa que escribamos entre comillas es una cadena. Algo que no dijimos es que las cadenas están hechas de piezas más pequeñas: los caracteres. Un caracter es simplemente una letra o símbolo.

2.1. Longitud

La función `len` devuelve la cantidad de caracteres de una cadena. Por ejemplo:

```
fruta = "banana"
longitud = len(fruta)
print(longitud)
```

Esto muestra un 6.

2.2. Recorrido

Algo común para hacer con una cadena es comenzar desde el principio, seleccionar cada caracter por vez, hacer algo con él, y continuar hasta el fin. Este patrón de procesamiento se llama **recorrido**. Una forma natural de codificar un recorrido es mediante una sentencia `for`:

```
for char in fruta:
    print(char)
```

Este ciclo recorre la cadena e imprime cada letra en líneas separadas. Cada vez que recorremos el ciclo, se asigna a la variable `char` el siguiente caracter de la cadena. El bucle continúa hasta que no quedan más caracteres.

Como ejercicio, escribí un programa que tome una cadena como parámetro y que imprima solo las consonantes de la cadena, en líneas separadas.

Notas preliminares

- Los ejercicios marcados con el símbolo ★ constituyen un subconjunto mínimo de ejercitación. Sin embargo, aconsejamos fuertemente hacer todos los ejercicios. Los ejercicios marcados con el símbolo ♣ presentan una mayor dificultad que el resto.

Parte 1 – Programas que muestran números

Ejercicio 1 ★

a) Hacer un programa que muestre, mediante un ciclo, los primeros 5 números naturales (1, 2, 3, 4 y 5).
b) Hacer un programa que permita al usuario elegir un número n y luego muestre los primeros n números naturales $(1, 2, \cdots, n)$.

Ejercicio 2 ★

a) Hacer un programa que muestre, mediante un ciclo, los números desde el 4 hasta el 7 (4, 5, 6 y 7).
b) Hacer un programa que permita al usuario elegir un número m y un n y luego muestre todos los naturales entre m y n $(m, m+1, m+2, \cdots, n-1, n)$. ¿Qué pasa si n es menor que m?

Ejercicio 3 ★

a) Hacer un programa que muestre, mediante un ciclo, los 5 números naturales que le siguen al 10 $(11, 12, \cdots, 15)$.
b) Hacer un programa que permita al usuario elegir un número n y luego muestre los 5 números naturales que le siguen a n $(n+1, n+2, \cdots, n+5)$.
c) Hacer un programa que permita al usuario elegir un número n y un número c, y luego muestre los c números naturales que le siguen a n $(n+1, n+2, \cdots, n+c)$.

Ejercicio 4 ★

a) Hacer un programa que muestre, mediante un ciclo, los números desde el 5 hasta el 11 salteando de a 2 elementos (5, 7, 9 y 11)
b) Hacer un programa que permita al usuario elegir un número m y un n y luego muestre todos los naturales entre m y n, pero salteando de a 3. Por ejemplo, si el usuario ingresara un n igual a 2 y un m igual a 14, el programa deberá mostrar `2, 5, 8, 11, 14`.
c) Hacer un programa que permita al usuario elegir un número n, un m y un p y luego muestre todos los naturales entre m y n, pero salteando de a p números. Por ejemplo, si el usuario ingresara un n igual a 2 y un m igual a 14, y un p igual a 4, el programa deberá mostrar `2, 6, 10, 14`.

Ejercicio 5 ★

a) Hacer un programa que muestre, mediante un ciclo, los números desde el 8 hasta el 3 $(8, 7, 6, 5, 4, 3)$.

Ejercicio 6 ★

a) Hacer un programa que muestre, mediante un ciclo, los números desde el 15 hasta el 6 pero salteando de a tres $(15, 12, 9, 6)$.

Ejercicio 7 ★

Hacer todos los ejercicios anteriores de nuevo, pero esta vez utilizando la sentencia **while** en lugar de **for**. De haberlos hecho con **while**, rehacerlos utilizando **for**.

Ejercicio 8

a) Hacer un programa que reciba un número n y muestre todas las potencias de 2 menores a n. Por ejemplo, si el usuario ingresa 20, el programa mostrará: 1 2 4 8 16. Ayuda: pensar primero si sería más práctico utilizar la sentencia **while** o **for**.
b) Hacer un programa que reciba un número n $(n > 0)$ y muestre las n primeras potencias de 2. Por ejemplo, si el usuario ingresa 6, el programa mostrará: 1 2 4 8 16 32.
c) Hacer un programa que reciba un número n $(n > 0)$ y muestre las n primeras potencias de n. Por ejemplo, si el usuario ingresa 4, el programa mostrará: 1 4 27 256. Es decir, 1^1 2^2 3^3 4^4.

Ejercicio 9

a) Hacer un programa que permita al usuario elegir un número positivo n y luego muestre en pantalla todos los divisores de n.
b) Hacer un programa que permita al usuario elegir un número positivo n y luego muestre en pantalla todos los divisores pares de n.
c) Hacer un programa que permita al usuario elegir un número positivo n y luego muestre en pantalla la cantidad de divisores de n.
d) Hacer un programa que permita al usuario elegir un número positivo n y luego muestre en pantalla la suma de los divisores de n.
e) Hacer un programa que permita al usuario elegir dos números positivos c y n y luego muestre en pantalla los primeros c divisores de n.
f) Hacer un programa que permita al usuario elegir dos números positivos c y n y luego muestre en pantalla los últimos c divisores de n.

Ejercicio 10

Hacer un programa que permita al usuario elegir un número positivo n y luego muestre en pantalla el producto (es decir, la multiplicación) de los numeros entre 1 y n.

Ejercicio 11

Hacer un programa que reciba un número m y determine el primer n para el cual la suma $1+2+...+n > m$. Por ejemplo, si el usuario ingresa 11 se deberá retornar 5 ya que $1+2+3+4 = 10 < 11$ y $1+2+3+4+5 = 15 > 11$

Ejercicio 12

a) Hacer un programa que permita al usuario elegir un número positivo n y luego muestre en pantalla los n primeros términos de la sucesión $a_n = 2n$. Es decir 2, 4, 6...
b) Idem anterior para la sucesión $a_n = 2n - 1$.
c) Idem anterior para la sucesión $a_n = n^2$.
d) Idem anterior para la sucesión $a_n = n^3 - n^2$.
e) Idem anterior para la sucesión $a_n = \frac{1}{n^2}$.

Ejercicio 13

a) Hacer un programa que permita al usuario elegir un número positivo n y luego muestre en pantalla las n primeras sumas parciales de la sucesión $a_n = 2n$. Es decir, 2 6 12 20...
b) Idem anterior para la sucesión $a_n = n^2$.
c) Idem anterior para la sucesión $a_n = n^3 - n^2$.
d) Idem anterior para la sucesión $a_n = \frac{1}{n^2}$.
e) ¿A qué valor se va acercando la suma del inciso anterior a medida que utilizamos un valor alto de n?

Ejercicio 14 ★

El logaritmo natural de 2 ($\ln(2)$) se puede aproximar de la siguiente manera:

$$\ln(2) = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} \dots$$

a) Escribir un programa que le pregunte al usuario la cantidad de términos a sumar y que muestre la aproximación de $\ln(2)$ con esa cantidad de términos.
b) ¿A partir de cuántos términos el valor alcanzado está a menos de 0.1 del valor que da la calculadora? En python se puede aproximar este valor usando math.log(2)
c) ¿A partir de cuántos términos el valor alcanzado está a menos de 0.01 del valor que da la calculadora?
d) Modificar el programa para que en lugar de pedir la cantidad de términos a sumar, pida al usuario un número decimal ϵ (muy chico) y calcule la suma hasta que el error comparado con el valor de la calculadora sea menor que ϵ

Ejercicio 15 ★

El número $\frac{1}{4}\pi$ se puede aproximar de la siguiente manera:

$$\frac{1}{4}\pi = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15}$$

a) Escribir un programa que le pregunte al usuario la cantidad de términos a sumar y que muestre la aproximación de π con esa cantidad de términos.
b) ¿A partir de cuántos términos el valor alcanzado está a menos de 0.1 del valor que da la calculadora?
c) ¿A partir de cuántos términos el valor alcanzado está a menos de 0.01 del valor que da la calculadora?
d) Modificar el programa para que en lugar de pedir la cantidad de términos a sumar, pida al usuario un número decimal ϵ (muy chico) y calcule la suma hasta que el error comparado con el valor de la calculadora sea menor que ϵ

Ejercicio 16 ★

Escribir un programa que solicite al usuario un número positivo y aproxime el valor del número e de la siguiente manera: (ejemplo para 7 términos)

$$\frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \frac{1}{6!}$$

a) ¿A partir de cuántos términos el valor alcanzado está a menos de 0.1 del valor que da la calculadora?
b) ¿A partir de cuántos términos el valor alcanzado está a menos de 0.01 del valor que da la calculadora?
c) Modificar el programa para que en lugar de pedir la cantidad de términos a sumar, pida al usuario un número decimal ϵ (muy chico) y calcule la suma hasta que el error comparado con el valor de la calculadora sea menor que ϵ

Ejercicio 17

a) Hacer un programa que permita al usuario elegir un número m y un n y muestre pares de numeros complementarios, o sea $(m,n)(m+1,n-1)(m+2,n-2)\ldots(n-1,m+1)(n,m)$. Por ejemplo, el usuario ingresa 5 y 10, 5 será el complementario de 10, 6 el de 9 y 7 el de 8, y deberá mostrarse:

```
5 10
6 9
7 8
8 7
9 6
10 5
```

b) Ídem anterior pero deberá frenarse cuando el lado izquierdo pase a ser más grande que el derecho.

Ejercicio 18 ★ ♣

a) Escribir un programa que permita al usuario elegir un número m y un n y muestre todos los pares de numeros que se pueden formar con los números que están entre ellos. Por ej. si el usuario ingresara 4 y 6, el programa deberá mostrar

```
4 4
4 5
4 6
5 4
5 5
5 6
6 4
6 5
6 6
```

b) Cambiar el programa para que use sólo un ciclo en vez de dos.

Ejercicio 19 ♣

a) Escribir un programa que permita al usuario elegir un número m y un n y muestre todos los pares de numeros que se pueden formar con los números que están entre ellos, pero esta vez que lo haga sin repetir inversos. Por ej. si el usuario ingresara 4 y 6, el programa deberá mostrar

```
4 4
4 5
4 6
5 5
5 6
6 6
```

b) Cambiar el programa para que use sólo un ciclo en vez de dos.

Ejercicio 20 ★

Hacer un programa que simule el juego de la quiniela. El usuario debe elegir un número del 0 al 99 y un monto a apostar, si acierta el número gana 70 veces lo apostado. (El número de la suerte debe ser elegido al azar)

Ejercicio 21 ★

Hacer un programa que permita al usuario jugar al piedra, papel o tijera contra la computadora. Se debe jugar al mejor de 5, es decir, si uno de los participantes consigue 3 victorias el juego termina.

Ejercicio 22 ★

¿Para qué números enteros distintos de cero es cierto que A + B + C = A x B x C ? (lo curioso es que sólo hay una solución)

Parte 2 – Cadenas

Ejercicio 23 ★

a) Escribir un programa que pida al usuario un número n y muestre una línea de n asteriscos. Ejemplo, para $n = 8$, el programa deberá mostrar:

```
********
```

b) Escribir un programa que pida al usuario un número n y muestre n líneas de $1, 2, 3, ...n$ asteriscos respectivamente. Ejemplo, para $n = 5$, el programa deberá mostrar:

```
*
**
***
****
*****
```

c) Escribir un programa que pida al usuario un número n y muestre n líneas de $2n - 1$ asteriscos respectivamente. Ejemplo, para $n = 5$, el programa deberá mostrar:

```
*
***
*****
*******
*********
```

Ejercicio 24 ★

a) Sabiendo que la pantalla de la consola tiene 80 caracteres de ancho, hacer un programa que, dada una palabra, la escriba en el centro de la pantalla.
b) Hacer una programa que, dada una palabra, la escriba pegada a la derecha de la pantalla.

Ejercicio 25 ★

Hacer una programa que, dada una palabra, la escriba recuadrada por asteriscos. Por ejemplo, si la palabra es "Ganaste", el programa debería escribir:

```
***********
* Ganaste *
***********
```

Ejercicio 26

Hacer un programa que dada una palabra y una letra, imprima la cantidad de apariciones de esa letra.

Ejercicio 27

Escribir un programa que pida al usuario una cadena e indique si esta posee un diptongo (dos vocales unidas).

Ejercicio 28

Escribir un programa que pida al usuario una cadena y una letra y reemplace las apariciones de dicha letra por asteriscos. Por ejemplo, si el usuario ingresa:
"*programador*"
"*r*"
el programa debe devolver "$p * og * amado*$"

Ejercicio 29

Hacer un programa que genere una clave provisoria para la gestión online de clientes de una empresa. El programa le pedirá el apellido y generará una clave de 5 caracteres, tomará las primeras 4 consonantes de la palabra ingresada. Cuando las consonantes no alcancen a 4, las faltantes las reemplazará por "*". Ejemplos:
clemente CLMN
rivera RVR*
oreo R***
La clave se completará con 1 dígito generado aleatoriamente entre 0 y 9.
Ejemplos: CLMN1 RVR*4 R***7

Ejercicio 30

Hacer un programa que solicite dos cadenas, nombre y apellido y arme el legajo de estudiantes de la universidad de la siguiente manera:
Los 3 primeros números del legajo coinciden con los primeros del dni luego "_", luego las 3 primeras letras del apellido y por ultimo la primera y ultima del nombre.
Por ejemplo:
JavierRodriguez 38946702
Legajo: 389_*rodjr*