Subprocesos

A medida que los algoritmos que analizamos son más complejos, encontramos que necesitamos repetir bloques de código iguales en diferentes partes de nuestros programas.

En ese caso nos convendría disponer de algún mecanismo que nos permitiese asignar un nombre a esos pedazos de código, establecer si fuera necesario condiciones iniciales, y usar esas instrucciones sin más que invocar ese nombre con sus correspondientes condiciones iniciales.

Por ejemplo, en ejercicios anteriores hemos implementado el código necesario para verificar si tres números dados, día, mes y año, forman una fecha válida.

```
Si día >=1 y mes >=1 y año >=0 entonces
      Si mes es 1, 3, 5,7, 8, 10, 12  entonces
            si día <=31 entonces
                  escribe(es fecha correcta)
            sino
                  escribe(fecha incorrecta)
      si no si mes es 4,6,9,11 entonces
                  si día <= 30 entonces
                        escribe(es fecha correcta)
                  sino
                        escribe(fecha incorrecta)
      si no si mes es 2
                  si año divisible por 4 y año no divisible por 400 entonces
                        si  día <= 29 entonces
                              escribe(es fecha correcta)
                        sino
                              escribe(fecha incorrecta)
                  si no
                        si  día <= 28 entonces
                              escribe(es fecha correcta)
                        sino
                              escribe(fecha incorrecta)
      si no
            escribe(fecha incorrecta)

sino
      escribe(fecha incorrecta)
```

Imaginemos, ahora, que tuviésemos que pedir tres fechas y que, para cada una, tuviésemos que escribir todas las instrucciones anteriores para validar cada fecha. El programa, evidentemente, tendría muchísimas líneas repetidas.

Para resolver esto, los lenguajes de programación disponen de mecanismos que permiten agrupar **bloques de código bajo un mismo nombre** y ejecutar esos bloques de instrucciones sin más que usar el nombre que le hemos asignado. A cada bloque de código representado de esta manera, lo llamaremos en nuestro pseudocódigo un **subproceso**, y la forma de declararlos será como sigue.

```
Subproceso nombre_subproceso([argumento [,argumento,….]]) [devuelve tipo_valor]
      <sentencias>
      [devuelve(valor devuelto [,valor devuelto])]
```

Así, por ejemplo, el caso de validación de una fecha, podríamos definir el siguiente subproceso.

```
Subproceso valida_fecha(dia, mes, año) devuelve valor lógico

      Si día >=1 y mes >=1 y año >=0 entonces
            Si mes es 1, 3, 5,7, 8, 10, 12  entonces
                  si día <=31 entonces
                        devuelve(fecha correcta-Verdad)
                  sino
                        devuelve(fecha incorrecta)
            si no si mes es 4,6,9,11 entonces
                        si día <= 30 entonces
                              devuelve(fecha correcta-Verdad)
                        sino
                              devuelve(fecha incorrecta-Falso)
            si no si mes es 2
                        si  año  divisible  por  4  y  año  no  divisible  por  400
                  entonces
                              si  día <= 29 entonces
                                    devuelve(fecha correcta-Verdad)
                              sino
                                    devuelve(fecha incorrecta-Falso)
                        si no
                              si  día <= 28 entonces
                                    devuelve(fecha correcta-Verdad)
                              sino
                                    devuelve(fecha incorrecta-Falso)
            si no
                  devuelve(fecha incorrecta-Falso)

      sino
            devuelve(fecha incorrecta-Falso)
```

Al proceso tenemos que darle todos los datos necesarios para que pueda hacer su trabajo. Por eso el proceso valida_fecha tiene como argumentos unas

variables día, mes y año sobre las que verificar si forman una fecha válida.

El subproceso devuelve resultados con la instrucción devuelve(<valor>), siendo <valor> el dato que diga si, en este caso, es una fecha valida o no.

Una vez declarado y definido este subproceso, podríamos escribir un programa como el que sigue.

```
escribe(“Dame el día:”)
leer(dia)
escribe(“Dame el mes:”)
leer(mes)
escribe(“Dame el año:”)
leer(año)
si valida_fecha(dia, mes,año) es fecha correcta entonces
     escribe(“la fecha es correcta”)
si no
     escribe(“la fecha es incorrecta”)
```

Si quisiésemos pedir otra fecha, deberíamos repetir las instrucciones escribe/lee anteriores y volver a validar la fecha. Podríamos, entonces, pensar en el siguiente subproceso.

```
Subproceso lee_fecha() devuelve tres enteros
     escribe(“Dame el día:”)
     leer(dia)
     escribe(“Dame el mes:”)
     leer(mes)
     escribe(“Dame el año:”)
     leer(año)
     devuelve dia, mes, año
```

El subproceso anterior lo podríamos usar de la siguiente manera.

```
dia1, mes1, año1 = lee_fecha()
dia2, mes2, año2 = lee_fecha()

si valida_fecha(dia1, mes1, año1) es fecha correcta y
   valida_fecha(dia2, mes2, año2) es fecha correcta  entonces

     escribe(“ambas fechas son correctas”)
sino
     escribe(“alguna fecha NO es correcta”)
```

Podríamos, también, definir el siguiente subproceso que muestra, con formato, una fecha por pantalla.

```
Subproceso escribe_fecha(dia, mes, año)
     escribe(“La fecha es: ”, dia,“/”, mes,“/”,,año)
```

Y, así, completar el algoritmo final como sigue:

```
dia1, mes1, año1 = lee_fecha()
dia2, mes2, año2 = lee_fecha()

si valida_fecha(dia1, mes1, año1) es fecha correcta y
   valida_fecha(dia2, mes2, año2) es fecha correcta  entonces

     escribe(“ambas fechas”)
     escribe_fecha(dia1, mes1, año1)
     escribe_fecha(dia2, mes2, año2)
     escribe(“son correctas”)

sino
     escribe(“alguna fecha NO es correcta”)
```

Debemos hacer notar las diferencias siguientes entre los procesos anteriores y alguna licencia tomada en nuestra definición de pseudocódigo ligada al lenguaje Python que usamos para la implementación.

En primer lugar, hay que notar que los subprocesos `valida_fecha` y `lee_fecha` devuelven ambos uno o varios valores, mientras que el subproceso `escribe_fecha` no devuelve ninguno.

Esta diferencia es importante y debemos considerarla siempre en el momento de diseñar un subproceso.

Además, el proceso `lee_fecha` no necesita de ningún argumento de entrada (ningún valor entre los paréntesis) pero devuelve tres valores, algo que no todos los lenguajes de programación permiten, pero que Python, como veremos después, sí. Esto nos simplifica esta introducción a la programación y

nos evita, por el momento, la necesidad de introducir conceptos como el **paso por valor o por referencia** o la definición de **estructuras de datos.** Qué significan y porqué y cómo lo evitamos aquí lo dejo a vuestra curiosidad. Por supuesto podéis preguntarme si Google no os resuelve dicha curiosidad...

A modo de resumen, podemos encontrar dos tipos de subprocesos:

- subprocesos, con argumentos o no, que no devuelven ningún valor. En estos casos, lo que realicen puede o no afectar al valor de las variables pasadas como argumento (si las variables son del tipo que admite modificación- detalle que es dependiente del lenguaje de programación que usemos-).
- subprocesos, con argumentos o no, que devuelven algún valor. En este caso, deberemos conocer en detalle el lenguaje de programación que utilicemos para saber cómo poder controlar y devolver los valores que interesen.

Python: defining subprocesses

And, what if do you want to reuse code in Python? You can asign a name to a piece of code just using the word `def`.

For example, let's define a subprocess called `greatest` thath, given two numbers, returns the greatest. Then, we should write the following code.

```
def greatest(a,b):
   if a>b:
     return(a)
   else:
     return(b)
```

Note that you need to pass both numbers, `a` and `b`, to be compared. These variables are called ***arguments***. Also, you must note that we use the instruction `return` to send, in this case, the greatest of both numbers to the point of the program where the function was used.

For example, the following file contains and use the `greatest` function defined before.

```
def greatest(a,b):
   if a>b:
     return(a)
   else:
     return(b)


########################
# Main program
########################

number1=int(input("Give me a number:"))
number2=int(input("Give me another one:"))

print("The greatest of", number1, "and ", number2, "is the ",
       greatest(number1,number2))
```

Of course, not all the subprocesses that we are going to need, must return a value. And, also we could need a function that need to return more than one value.

Python allow to define, without being necessary to declare in any way, functions of any type.

Imagine that you just need a subprocess to show on screen a menu as, for example, options of a calculator; and you could also define the function that, depending on the numbers and the operation you want to do, it returns the result of this operation.

Then, you could write the following code.

```
def showMenu():
   print("**************MENU***********")
   print("1.- SUM")
   print("2.- SUBSTRACT")
   print("3.- MULTIPLY")
   print("4.- DIVIDE")
   print()
   print("5.- EXIT")
   print("Select option: ")


def operate(a,b,op):
   if   op==1:
      return(a+b)
   elif op==2:
      return(a-b)
   elif op==3:
      return(a*b)
   elif op==4:
      return(a/b)

########################
# Main program
########################

showMenu()
numOp=int(input())
```

```
while numOp!=5:

   num1=int(input("Enter a number:"))
   num2=int(input("Give me a number:"))

   print("RESULT:", operate(num1,num2,numOp))

   showMenu()
   numOp=int(input())

print("Bye and thanks by using this calculator...")
```

As you can see, the code of the main program is more legible than if we had written the instructions to print the menu, and the ones to make the ops inside the `while` sentence.

Finally, Python allows that functions return more than one value. Let's see the code to read a date.

```
def read_date():

     day=int(input("Day: "))
     month=int(input("Month: "))
     year=int(input("Year: "))

     ##VALIDATION OF DATE
     …
     …
     return day, month, year


Use at Main program:

     ...
     day1,month1,year1=read_date()
     day2,month2,year2=read_date()
     …
```

That avoids us the problem of reference/value arguments and, also, to define structures of data to be returned by procedures.

To introduce yourself in defining functions, you should implement those indicated on the activities document. Most of them are algorithms that you have programmed before. You just need to transform them into functions to be used in a main program.

It's easy, let's start!