El futuro digital es de todos

MinTIC

# Control de flujo: Patrón, Ciclos

Universidad de Caldas

# Hola a todos,

cuando construimos software podemos hacer uso de diseños ya existentes,  estrategias ya planteadas, de algoritmos probados y de programas que nos van a facilitar este gran proceso creativo que es el desarrollo de software, de eso se trata el tema de hoy.

Un principio de programación, que permite “no reinventar la rueda”  puede ayudarnos a que la programación se vuelva más efectiva al reutilizar elementos que ya han sido analizados y probados por otros, de ahí este principio que vamos a ver hoy se relaciona con el tema de patrones.

Revisemos la importancia de los patrones en la programación, supongamos que hay una primera idea que se plantea para solucionar un problema, no olvidemos que a pesar de la juventud que tiene el desarrollo de software con respecto a otras profesiones se han creado muchas estrategia que podemos reutilizar adaptándolas a un contexto específico para que puedan apoyar la solución que se pretende presentar. Esto tiene un nombre muy técnico conocido como patrón, el cual es una solución reutilizable a un problema común en un contexto específico, hay que tener en cuenta que estos patrones normalmente no son algoritmos al pie de la letra, que en situaciones específicas nos puede permitir recordar el patrón y volverlo a aplicar.

Estudiemos a continuación, unos patrones que tiene que ver con un tipo de recorridos, esto se encuentra asociado al concepto de patrones con ciclos, y es que muchas veces cuando hacemos un ciclo while se tiene claramente el lugar donde éste empieza y dónde termina, es decir, de antemano sabemos cuántas veces se va a repetir un conjunto de acciones, aquí vamos a agrupar un conjunto de patrones que cuenta con estas características, es decir se conoce el número de veces que se va a repetir algo.

**Algoritmo:** Patrón contador

1: **Inicializar** contador
2: **Mientras** contador no llegue al limite
3: #instrucciones
4: **incrementar** contador
5: #instrucciones

Lo anterior significa, se controla el límite del contador e incrementa el contador, que lo normal es que se incremente de 1 en 1 sin embargo no es obligatorio este concepto.

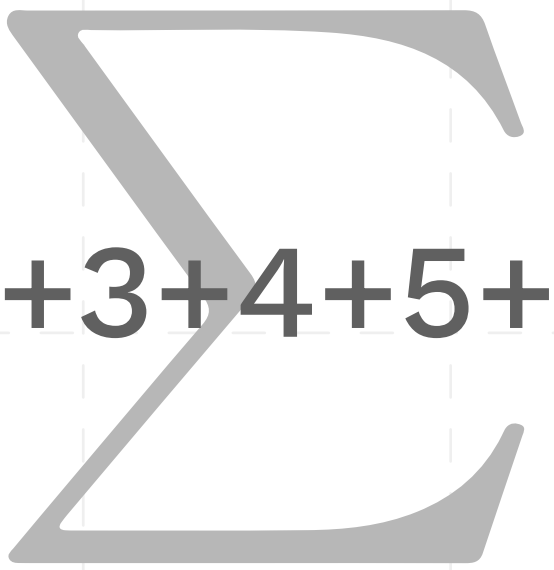
Analicemos entonces los siguientes ejemplos:

```
def conde_contador (limite)
Cont=1 #Inicializa el contador
Whilie cont <= limite:
Print (cont)
Cont+=1 #incrementa el contador
Conde_contar (5)
```

profundizando en este ejemplo, está presente la definición de la función def_conde_contar, el límite, inicializamos el contador en 1, luego controlamos que no llegue al final del límite imprime el contador e incrementa el contador. Se inicializa.

```
def conde_contador (5)
Cont=1 #Inicializa el contador
Whilie 6 <= 5: False
Print (cont)
Cont+=1 #incrementa el contador
Conde_contar (5)
```

Se realiza, hasta que sea falsa y termine la ejecución, regresando al inicio.
Este es un caso típico de un algoritmo para contar, pero ¿cómo aumentamos de 2 en o de 3 en 3 ?
lo podemos lograr con el  patrón sumatorio, una sumatoria es la suma consecutiva de un conjunto de números.



$$S: 1+2+3+4+5+6+7...$$

En este punto se está programando algo similar a: “a esa suma S llévele un 1 +2+3+4+5" y así sucesivamente, es importante analitico y pensar, “esto no es el patrón de conteo pero es muy parecido”,  de ahí la importancia de adaptarse a los patrones.

---

**Algoritmo:** Patrón sumatoria

---

1: **Inicializar** contador
2: **Inicializar** sumatoria
3: **Mientras** contador no llegue al limite
4:  #instrucciones
5: **incrementar** contador
6: **actualizar** sumatoria
7:  #instrucciones

En el siguiente caso se inicializa el contador pero en alguna parte también se debe  inicializar la suma y mientras el contador no llegue al límite debemos incrementar el contador y actualizar la sumatoria. Veamos a continuación  un ejemplo en Python

```
def calcular_sumatoria (limite)
    Cont=1 #Inicializa el contador
    sumatoria= 0 #inicializa la suma en 0
    Whilie cont <= limite:
        sumatoria=sumatoria+cont #Realiza la sumatoria
        Cont+=1 #incrementa el contador
    return sumatoria
total_sumatoria=calcular_sumatoria (3) # 1+2+3
```

---

```
def calcular_sumatoria (3)
    Cont=1 #Inicializa el contador
    sumatoria= 0 #inicializa la suma en 0
    Whilie 4 <= 3 :
        sumatoria=sumatoria+cont #Realiza la sumatoria
        Cont+=1 #incrementa el contador
    return 6
total_sumatoria=calcular_sumatoria (3) # 1+2+3
```

En esta parte del código, se quiere programar la sumatoria indicando al programa que lleve lo que tenia antes mas lo que vale el contador y eso actualiza el valor, por lo tanto la próxima vez que ingresa al sitio ya tiene el nuevo valor de la sumatoria y vuelve y lo actualiza; esto sencillo, se recomienda un concepto que se llama **prueba de escritorio**, pregunten a su tutor o formado lo que significa esto para puedan hacer más ejercicios y validar su desempeño.

en este momento llevamos dos patrones, el patrón contador y el patrón sumatorio, aquí viene un tercer patrón que es el patrón acumulador.

**Algoritmo:** Patrón acumulador

1: **Inicializar** contador
2: **Inicializar** acumulador
3: **Mientras** contador no llegue al limite
4: #instrucciones
5: **actualizar**contador
6: **incrementar** sumatoria
7: #instrucciones

De nuevo inicializamos el contador y el acumulador, mientras el contador no llegue al límite, actualizamos el acumulador e incrementamos el contador. El asunto con el acumulador es que ya no es como la sumatoria (que es una serie consecutiva de números), sino, que nos permite hacerlo con cualquier valor diferente como sumar los valores de una factura, veamos:

```
def calcular_total_pago (numero_productos):
    Cont=1 #Inicializa el contador
    total_pago= 0 #el total a pagar
    Whilie cont <= numero_productos:
        precio=float (imput(”digite el precio del producto:”))
        total_pago=total_pago+precio va acumulando los
precios
        Cont+=1 #incrementa el contador
    return total_pago

calcular_total_pago (2)
```

```
def calcular_total_pago (2):
    Cont=1 #Inicializa el contador
    total_pago= 0 #el total a pagar
    Whilie 3 <= 2
        precio=float (imput(”digite el precio del producto:”)
        total_pago=10 + 90 va acumulando los precios
        Cont+=1 #incrementa el contador
    return 100

total_a_pagar=calcular_total_pago (2)
```

Este es otro patrón si observamos con precaución son muy parecidos a los anteriores y pueden surgir muchos más, todos en el mismo esquema. así se puede ir adaptando cosas y no es necesario “reinventar la rueda como” dijimos al principio.

Otro conjunto de patrones pero con ciclos, la diferencia es que no se sabe en qué momento termina el ciclo, desconocemos el número de veces que se va a repetir algo, y no siempre que se ejecute se va  a repetir las mismas veces.

Veremos un patrón de estos que no conocemos el número de veces que se va a repetir y lo vamos a llamar el patrón de decremento y es lo contrario al acumulador, en lugar de incrementar empieza a decrementar, de esta manera podemos analizar el tema de los ciclos que no conocemos el número de veces que se va a repetir con este patrón.

**Algoritmo:** Patrón decremental

1: **Inicializar** limite inferior
2: **Mientras** no se llegue al limite inferior
3: #instrucciones
4: **actualizar** limite inferior
5: #instrucciones

Inicializamos un límite inferior, y mientras no llegue al límite inferior, se le indica al programa que actualice el límite inferior, debería estar restando o dividiendo cualquier cosa que vuelva más pequeño el valor de ese límite inferior, aquí lo vemos precisamente con dinero supongamos que queremos gastar un dinero y queremos saber el total de dinero que voy a gastar y queremos ir conociendo a medida que voy comprando productos cuánto dinero me va quedando.

```
def motrar_dinero_disponible (total_dinero):

    Whilie total_dinero > 0:
        precio=float (imput(”digite el precio del producto:”))
    if precio<=total_dinero: #verifica que puede comparlo
        total_dinero=total_dinero - precio #va actualizando el saldo
    else:
        print(”no puede comprar ese producto”)
    print (” aún te quedan $”, total_dinero,”para gastar”)

    return total_dinero
saldo= mostrat_dinero_disponible (100000)
```

```
def motrar_dinero_disponible (100000):

    Whilie 0 > 0:
        precio=float (imput(”digite el precio del producto:”))
    if 40000 <= 40000: #verifica que puede comparlo
        total_dinero= 40000 - 40000 #va actualizando el saldo
    else:
        print(”no puede comprar ese producto”)
    print (” aún te quedan $”, 0,”para gastar”)

    return total_dinero
saldo= mostrat_dinero_disponible (100000)
```

Preguntale a tu tutor o formador, si puede ser que alguna vez retorne algo mayor a 0 o me toca gastarme hasta que ya no tenga nada, analiza bien esto.

Hemos llegado al final de este video en revista, en éste hemos visto cosas bastante interesantes, nuestro segundo principio de programación que es no reinventar la rueda porque ya hay muchos algoritmos que funcionan, lo que debemos hacer es saberlos buscar, cuando se encuentran con el título de patrón como los de ciclos vamos más a la fija, vimos el patrón contador, sumatorio y acumulador estos 3 están controlados y sabíamos el número de veces que se hace el ciclo while.

También vimos otro, el decremento que básicamente fue un ejemplo para los casos que vamos a utilizar el while, en este sabemos que existen cosas que se repiten, pero no se el número de veces que se van a  repetir, básicamente esas son las dos cosas que debes saber del ciclo y con eso puedes hacer muchas cosas, con IF, con listas y ciclos se puede hacer software muy interesante, hemos dado muy buenos pasos en este módulo.

¡Hasta una próxima oportunidad!


Universidad de Caldas