



CICLO 1

[FORMACIÓN POR CICLOS]

Fundamentos de **PROGRAMACIÓN**



**UNIVERSIDAD
DE ANTIOQUIA**

Facultad de Ingeniería

Lectura

CONCEPTO E IDENTIFICACIÓN

de subprograma



En el desarrollo de soluciones, utilizando el computador como herramienta, se presentan problemas muy complejos. Una de las principales técnicas para solucionar estos problemas es identificar las diferentes tareas que hay que ejecutar y resolver cada una de ellas en forma independiente, para luego construir la solución ensamblando como un todo las tareas que se solucionaron independientemente. El programa correspondiente a cada una de esas tareas se denomina subprograma. Trataremos en este módulo el concepto de **subprograma** y un ejemplo que lo ilustre.

Definición.

Los subprogramas son algoritmos que se elaboran en forma independiente y que tienen la característica de que se pueden invocar desde cualquier programa cuando se necesiten. El objetivo principal de utilizar subprogramas es elaborar algoritmos más cortos, menos complejos, más legibles y más eficientes.

Ejemplo de uso.

En el tema de análisis combinatorio (conteo) es importante determinar el número de combinaciones que se pueden construir con **n** elementos, tomados en grupos de a **r** elementos.

Por ejemplo, si tenemos tres elementos **a, b** y **c**, y queremos formar combinaciones con dos de esos tres elementos, las posibles combinaciones son: **ab, ac** y **bc**.

El total de combinaciones de tres elementos (**n == 3**), tomando de a dos elementos (**r == 2**), es 3.

Si el número de elementos es cuatro (**n == 4**): **a, b, c** y **d**, las combinaciones posibles tomando de a dos elementos (**r == 2**) son: **ab, ac, ad, bc, bd** y **cd**, es decir, seis posibles combinaciones.

Para determinar el total de combinaciones de **n** elementos tomados en grupos de a **r** se tiene una fórmula matemática que se escribe así:

$${}^n C_r = \frac{n!}{r! (n-r)!}$$

y se lee así: el total de combinaciones de **n** elementos tomados en grupos de a **r** elementos es el factorial de **n** dividido por el producto del factorial de **r** con el factorial de **n - r**.

Si nuestro objetivo es elaborar un algoritmo en el cual se lean los datos de **n** y **r** y calcular el número de combinaciones que se pueden construir, debemos definir una variable para el factorial de **n**, otra variable para el factorial de **r** y otra para el factorial de **n - r**. Llamemos a estas variables **fn**, **fr** y **fnr**.

Construyamos un algoritmo para ejecutar esta tarea: leer dos datos **n** y **r** y calcular e imprimir el total de combinaciones que se pueden construir con **n** elementos tomados en grupos de a **r** elementos.

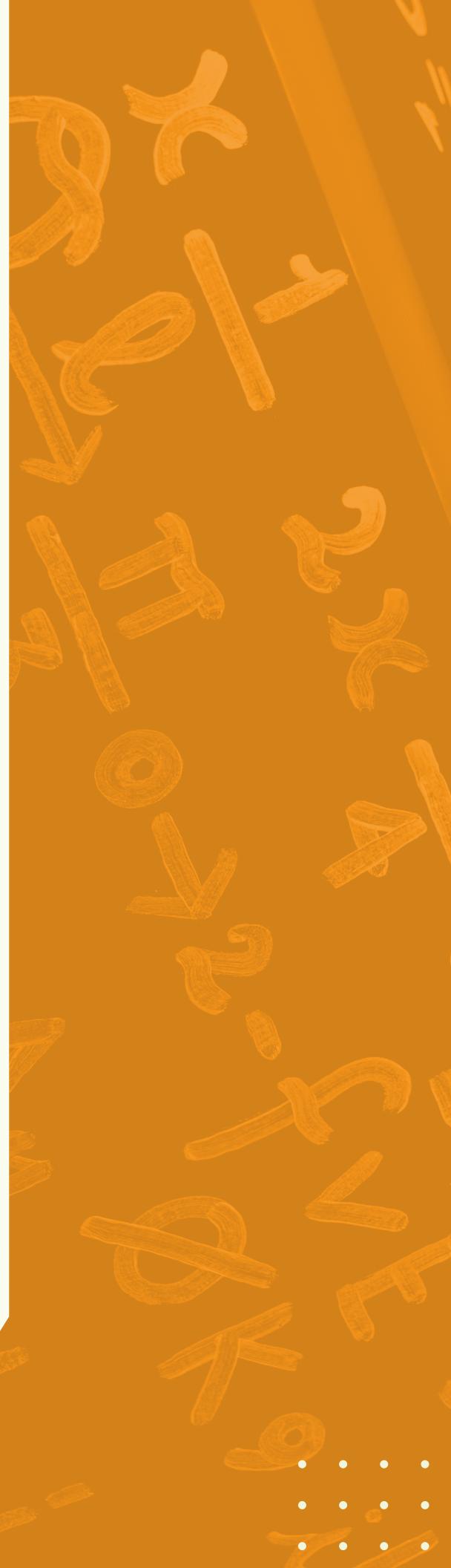
```
n = int(input("Entre valor de n "))  
r = int(input("Entre valor de r "))
```

```
fn = 1 # fn es la variable donde se  
        # almacenará el factorial de n  
for i in range(1, n + 1): # instrucciones para  
    fn = fn * i # calcular el factorial de n
```

```
fr = 1 # fr es la variable donde se  
        # almacenará el factorial de n  
for i in range(1, r + 1): # instrucciones para  
    fr = fr * i # calcular el factorial de r
```

```
fnr = 1 # fnr es la variable donde se  
        # almacenará el factorial de n - r  
for i in range(1, n - r + 1): # Instrucciones para calcular el  
    fnr = fnr * i # factorial de n - r
```

```
nc = fn // fr // fnr  
print(n, r, fn, fr, fnr, nc)
```





Como podrá observar, hay un grupo de instrucciones que se repiten tres veces. Son exactamente las mismas tres instrucciones pero con nombres de variable diferentes: **fn**, **fr** y **fnr**.

Si tuviéramos un mecanismo con el cual pudiéramos calcular el factorial de un número entero cualquiera, nuestro algoritmo quedará así:

```
n = int(input("Entre valor de n "))
r = int(input("Entre valor de r "))
fn = factorial(n)
fr = factorial(r)
fnr = factorial(n - r)
nc = fn // fr // fnr
print(n, r, fn, fr, fnr, nc)
```

donde **factorial(x)** es un subprograma que calcula el factorial de un entero **x** y retorna dicho valor. Así, en la primera instrucción calcula el factorial de **n**, retorna su valor y lo almacena en la variable **fn**; en la segunda instrucción calcula el factorial de **r**, retorna su valor y lo almacena en la variable **fr**; en la tercera instrucción calcula el factorial de **n - r**, retorna su valor y lo almacena en la variable **fnr**.

Como podrá observar, nuestro algoritmo es más corto y más legible. Ese programa se denomina "programa llamante" o "programa principal" y se almacena en un archivo. Llamemos a ese archivo **numeroCombinaciones**.

Ahora, ¿dónde se almacena el programa (que de aquí en adelante seguiremos llamando subprograma) en el que se calcula el factorial de un entero **x** cualquiera?

Una forma de hacerlo es en el mismo archivo en el que se tiene el programa principal. Entonces en nuestro archivo **numeroCombinaciones** quedará lo siguiente:



```
def factorial(n):
    f = 1
    for i in range(1, n + 1):
        f = f * i
    return f

n = int(input("Entre valor de n "))
r = int(input("Entre valor de r "))
fn = factorial(n)
fr = factorial(r)
fnr = factorial(n - r)
nc = fn // fr // fnr
print(n, r, fn, fr, fnr, nc)
```

Observe que el subprograma llamado **factorial** está precedido de la palabra **def**. Esta es una palabra reservada de Python que indica que a continuación viene el nombre de un subprograma, el cual entre paréntesis tendrá definidos los datos con los que debe trabajar. A estos datos se los denomina **parámetros**. Después del cierre de paréntesis siempre habrá que poner dos puntos.

Con base en lo anterior en nuestro programa hemos definido un subprograma que se llama factorial, recibe como entrada un dato que denominamos **x**, y retorna un valor que es el resultado de calcular el factorial de **x**. De esta forma podremos utilizar dicho subprograma en el programa principal.

OJO: todo subprograma se tiene que escribir antes de cualquier programa que lo utilice. En general, uno puede definir tantos subprogramas como desee.

Sin embargo, definir subprogramas en el archivo en el cual uno los utiliza no es lo más apropiado, dado que dichos subprogramas sólo podrán ser utilizados por el programa de ese archivo. Si nosotros almacenamos los subprogramas en un archivo aparte, podremos hacer uso de esos subprogramas en cualquier programa que se desarrolle.

Vamos entonces a construir un archivo en el cual sólo tengamos definidos subprogramas. Llaremos ese archivo "misFunciones".

Entonces, si tenemos almacenado el subprograma **factorial(x)** en un archivo llamado **misFunciones**, nuestro programa en el archivo **numeroCombinaciones** quedará así:

```
import misFunciones

n = int(input("Entre valor de n "))
r = int(input("Entre valor de r "))
fn = misFunciones.factorial(n)
fr = misFunciones.factorial(r)
fnr = misFunciones.factorial(n - r)
nc = fn // fr // fnr
print(n, r, fn, fr, fnr, nc)
```

Fijémonos que para poder utilizar el subprograma **factorial(x)** debemos poder acceder al archivo en el que se halla el subprograma **factorial(x)**. Dicho acceso se logra con la instrucción **import**.

La forma general de esta instrucción es simplemente la palabra **import** y el nombre del archivo en el cual se encuentra el subprograma. Luego, para utilizar el subprograma **factorial(x)**, debemos invocarlo usando el nombre del archivo que se importó, un punto y el nombre del subprograma.

Una forma abreviada de escribir el programa del archivo **numeroCombinaciones** es como sigue:

```
import misFunciones as mf
n = int(input("Entre valor de n "))
r = int(input("Entre valor de r "))
fn = mf.factorial(n)
fr = mf.factorial(r)
fnr = mf.factorial(n - r)
nc = fn // fr // fnr
print(n, r, fn, fr, fnr, nc)
```

Es decir, podemos importar el archivo asignándole un alias. En nuestro caso, el alias es **mf**.

En resumen, para escribir subprogramas debemos tener en cuenta cuál es la tarea que debe efectuar el subprograma, cuáles son los datos que se requieren para ejecutar dicha tarea y qué datos debe retornar el subprograma.

Los datos que se requieren para ejecutar un subprograma reciben el nombre de parámetros del subprograma. En nuestro ejemplo **factorial(x)** el parámetro es la variable **x**. Cuando se invoca **factorial(n)**, la variable **x** toma el valor de **n** y ejecuta las instrucciones del subprograma y retorna el factorial de **n**. Cuando se invoca **factorial(r)**, la variable **x** toma el valor de **r** y ejecuta las instrucciones del subprograma y retorna el factorial de **r**.

En general, todo subprograma debe retornar un valor; sin embargo, hay situaciones en las cuales solo se requiere que haga una tarea sin retornar ningún valor.

Por último, digamos que en Python los subprogramas reciben el nombre de **funciones**.

Variables locales y variables globales.

Dentro de un subprograma se pueden definir nuevas variables. Las variables que se definen dentro del subprograma se denominan *locales* y solo existirán mientras se esté ejecutando el subprograma; cuando termina la ejecución del subprograma, dichas variables desaparecen. En nuestro ejemplo de la función **factorial(x)**, las variables **f** e **i** son variables locales.

Si dentro de un subprograma se utilizan variables definidas por fuera del subprograma, dichas variables se llaman *globales*. En Python, si se desea trabajar variables globales, hay que especificar dentro del subprograma cuáles son esas variables globales.

Por ejemplo, si se tiene una función con las siguientes instrucciones:

```
def misterio(m, n):
    global p
    print("Misterio ")
    i = 1
    while i <= m:
        j = 1
        p = p + 1
        while j <= n:
            k = i * j
            print("\t", i, "\t*", "\t", j, "\t=", "\t", k)
            j = j + 1
        i = i + 1
    print(p)
```

La variable **p** está definida como global; por tanto, en todo programa principal o programa llamante que desee utilizar dicha variable **p** también debe estar definida como global. Esto significa que todas las modificaciones que se hagan a la variable **p** dentro de la función afectan el valor de **p** fuera de la función.