PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN
IIC1103 INTRODUCCIÓN A LA PROGRAMACIÓN

Ejercicios Propuestos sobre Funciones

Rodrigo Toro Icarte

General

Las funciones permiten definir un nombre para un trozo de código. Pueden recibir parámetros y retornar un valor. Para usarlas debes:

- **Definirlas**: Es decir, programar su comportamiento.
- **Llamarlas**: Es decir, invocarlas para que su código se ejecute.

Para definir una función se utiliza la siguiente sintaxis:

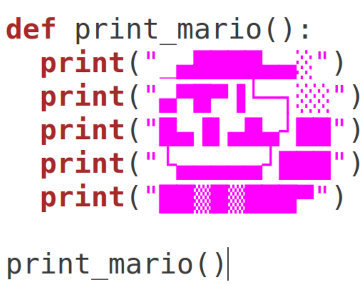
```
def nombre_función(param_1, param_2, ...):
    inst_1
    ...
    inst_n
    return ret
```

Para llamarla, simplemente se escribe el nombre de la función y se le da valor a sus parámetros:

```
out = nombre_función(in_1, in_2, ...)
```

Funciones sin parámetros ni retornos

La siguiente es una función muy simple. No recibe parámetros ni retorna valores. Simplemente imprime a Mario cada vez que se llama.



Ejercicio: Revisa el código `serpiente.py` disponible en el siding. Ese código muestra en consola 5 veces una serpiente. Mediante el uso de funciones redúcelo de 64 a 19 líneas de código.

Funciones con parámetros, pero sin retorno

Cada función define su propio espacio de variables (scope). Esto significa que solo pueden utilizar las variables que dentro de ella se definan. Sin embargo, muchas veces una función necesitará valores externos para

trabajar. En tales casos, dichos valores se entregan mediante los parámetros de la función. Por ejemplo:

```
# Imprime el máximo
def maximo(num1,num2):
  if(num1 < num2):
    print(num2)
  else:
    print(num1)

a = int(input("a: "))
b = int(input("b: "))
maximo(a,b)
```

Esta función imprime el máximo entre 2 números. Notar que los números no son definidos dentro de la función, le llegan en forma externa. Cuando llamamos a la función debemos asignar valores a cada uno de sus parámetros (en este caso, a `num1` y `num2`).

Ejercicio: En tu código modificado de `serpiente.py` implementa la función `mostrar(n)`. Esta función debe mostrar `n` veces la serpiente en consola. Para ello necesitarás un loop y llamar a tu función para mostrar la serpiente. Finalmente, desde tu código principal ejecuta `mostrar(5)`, que debería *printear* 5 veces la serpiente.

Funciones con parámetros y retorno

Además de recibir parámetros, las funciones pueden retornar valores. Esto nos permite usar el valor retornado desde fuera de la función. Por ejemplo, la siguiente función retorna el máximo entre 2 números.

```
# Imprime el máximo
def maximo(num1,num2):
  if(num1 < num2):
    return num2
  else:
    return num1

# Imprimo el máximo entre 4 números
a = int(input("a: "))
b = int(input("b: "))
c = int(input("c: "))
d = int(input("d: "))
m1 = maximo(a,b)
m2 = maximo(c,d)
print(maximo(m1,m2))
```

Gracias a que la función retornó el máximo, desde el código principal podemos trabajar con ese valor. Por ejemplo, podemos calcular cuál es el máximo entre 4 números.

Ejercicio: Implementa la función `es_primo(n)` que recibe un número natural `n` y retorna `True` ssi `n` es primo. Luego utilízala para encontrar la suma de los primeros `m` números primos (con `m` dado por el usuario). Por ejemplo, si `m` es `5`, entonces se debe mostrar 28 (2+3+5+7+11).

Ejercicios propuestos

1) Sumar los múltiplos de 3 o 5

A tu profesor le gustaría saber cuál es la suma de los múltiplos de `3` o de `5` menores que `n`. Por ejemplo, para `n=10` la suma es 23 (3+5+6+9). Debido a que debo terminar mi tesis para el 22 de Abril (tengo poco tiempo) por favor resuelve este problema por mí :). La estrategia es la siguiente:

- Crea una función que retorne la suma de los múltiplos de `j` menores que `n` (`j` y `n` son parámetros).
- Llama a tu función para resolver el problema.

Notar que la suma pedida se puede expresar como:

$$s = 3 \cdot \sum_{i=1}^{n//3} i + 5 \cdot \sum_{i=1}^{n//5} i - 3 * 5 \cdot \sum_{i=1}^{n//(3*5)} i$$

... donde n es el valor límite. Además recuerda que:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

2) Sumar dígitos de n!

Otra duda matemática que tengo es conocer el valor de la suma de los dígitos de 1000!. Recuerda que la definición matemática del factorial es:

$$n! = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n = \prod_{i=1}^{n} i$$

Por ejemplo, $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$, y la suma de sus dígitos es 3. En este caso, ya programé el código principal:

```
print(sumar_digitos(factorial(1000)))
```

... pero me dio lata programar las funciones. ¿Eres capaz de implementar `factorial(n)` y `sumar_digitos(n)`?

3) Coeficientes polinomiales

Crea un programa que calcule el coeficiente polinomial $C(m, n)$, que se define como:

$$C(m, n) = \frac{m!}{(m-n)!n!}$$

Donde los valores de `m` y `n` son enteros definidos por el usuario. Puedes reutilizar funciones que hayas creado en los ejercicios anteriores.

4) Examen 2014-2

El Gimnasio "Full Sport" define sus precios en base a 4 categorías de servicios: *básico*, *premium*, *super premium* y *complementario* (gratis para todos los clientes). Cada cliente tiene una tarjeta donde carga un monto a gastar y desde donde se le descuentan los servicios que utiliza de acuerdo al precio del servicio y ciertos descuentos. Estos descuentos dependen de la categoría del cliente al que pertenece (descuento porcentual) y de la edad del cliente (descuento bruto, que se aplica luego del descuento porcentual). Si un cliente no tiene suficiente carga en su tarjeta, no se le realiza el cobro, ya que no se le permite usar el servicio. Se te pide escribir el código correspondiente a 6 funciones que facilitarán realizar los cobros.

Cuadro 1: Valores de servicios

Servicio	**Valor**
básico	$ 10.000
premium	$ 25.000
super premium	$ 45.000
complementario	$ 0

Cuadro 2: Descuentos porcentuales

Categoría	**Descuento**
400	50 %
300	25 %
200	10 %
otros	0 %

Cuadro 3: Descuentos por edad

Edad	**Descuento**
Menor a 18 o 60 o más	$ 5.000
Entre 18 y 30	$ 1.000
otros	$ 0

Código a completar:

```
# Retorna el valor base de un servicio
def valor_servicio(tipo_servicio):

# Retorna el descuento proporcional por categoría.
# Por ejemplo si el descuento es del 25% debe retornar 0.25
# La categoria viene dada como string.
def descuento_proporcional_por_categoria(categoria):

# Retorna el descuento bruto por edad
def descuento_bruto_por_edad(edad):

# Retorna el valor de cobro por servicio incluyendo descuentos
def valor_cobro(tipo_servicio, edad, categoria):

# Realiza cobro según tipo de servicio, y retorna el saldo restante en la tarjeta
# Aplica los descuentos correspondientes y despliega mensaje indicando
# cuánto cobró a la persona identificada por su nombre
def realizar_cobro(tipo_servicio, nombre_cliente, saldo, edad, categoria):

# Despliega mensaje indicando cuántas veces puede consumir una persona
# (identificada por su nombre) cada uno de los servicios no gratuitos
# Por ejemplo, una persona con saldo 50.000 de 20 años y de categoria "200"
# podría consumir 6 veces el servicio básico, 2 veces el premium
# y 1 vez el super premium
def consumos_posibles(nombre_cliente, saldo, edad, categoria):
```

5) Números capicúa

Este problema es un clásico de intro. Crea una función que retorne True ssi un número `n` (parámetro) es capicúa. Los números capicúa son aquellos que se escriben igual en ambos sentidos. Por ejemplo 8, 313 y 1234321 son capicúa, pero 71, 123 y 1234322 no lo son.

6) Funciones en la tarea 1

En tu tarea 1 implementa (y utiliza) las siguientes funciones:

- `mostrar_tablero()`: Que despliega el tablero actual en consola.
- `es_valido(i,j,v)`: Que retorna `True` ssi el usuario puede poner en la casilla `(i,j)` el valor `v`.

7) Funcionalidad extra tarea 1

Originalmente la tarea 1 tenía una funcionalidad extra, llamada *"cheating"*. Esa era, probablemente, la parte más interesante de la tarea. Si el usuario seleccionaba esa opción, el programa debía buscar alguna casilla segura y marcarla. Las casillas seguras son aquellas para las que solo existe un valor posible (que no rompe las reglas del sudoku). De esta forma, mediante *"cheating"*, se podían completar los tablero 1 y 2 en forma automática. El tablero 3 no se podía completar porque, eventualmente, se llega a un estado en el que no hay más casillas seguras (todas tienen 2 o más valores posibles). Agrega esta funcionalidad a tu tarea.
Hint: Tener implementada la función `es_valido(i,j,v)` simplificará mucho tu código.