

UNIVERSIDAD DE ALCALÁ

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN



LABORATORIO
FUNDAMENTOS DE LA PROGRAMACIÓN
Tema 2

TEMA 2: FUNCIONES Y PROCEDIMIENTOS

1. FUNCIONES

En el tema anterior hemos aprendido a utilizar algunas de las funciones predefinidas por el lenguaje, como por ejemplo la función **round()**, que es una función predeterminada que no necesita invocar a ningún módulo para su correcto funcionamiento. En cambio, existen funciones que para poder invocarlas necesitamos previamente llamar a la librería que las contiene, por ejemplo para poder llamar a ciertas funciones matemáticas, como la raíz cuadrada **sqrt()**, tienen que ser importadas desde el módulo **math** (este módulo contiene muchísimas funciones de este tipo). La sintaxis que se utiliza para importar funciones es la siguiente:

```
from nombre_modulo import nombre_funcion
```

Para tener accesibles cualquiera de las funciones que contiene el módulo, basta con sustituir por un **asterisco (*)** el nombre_función. Por ejemplo, para acceder a todas las funciones del módulo **math** habría que escribir la siguiente instrucción: **from math import ***

Una función es un bloque de código el cual puede ser ejecutado cuando y cuantas veces deseemos. Ya hemos utilizado funciones predefinidas, ahora aprenderemos a crear nuestras propias funciones.

1.1. DEFINICIÓN Y USO DE SUBPROGRAMAS.

Las subrutinas en Python tienen el nombre de **funciones**, se tratan de un bloque de código que realiza una tarea específica y puede ser llamado las veces que sean necesarias.

Los subprogramas, son de gran ayuda cuando necesitamos que un programa ejecute una operación varias veces. Deben ser definidas en el bloque declarativo, es decir, antes de las instrucciones que componen el cuerpo principal del programa.

Sintaxis para definir funciones en Python:

```
def nombre_función (lista_de_parámetros):  
    sentencias
```

En una definición de función la palabra reservada del encabezado es **def** seguida del **nombre del subprograma (identificador)** y entre paréntesis la **lista de parámetros**. La lista de parámetros puede estar vacía o contener varios parámetros. En cualquier caso los **paréntesis** son necesarios, para finalizarlos añadiremos dos puntos.

La estructura es la siguiente:

- **Encabezado** → Empieza con la palabra reservada **def** y acaba con dos puntos (:).
- **Cuerpo del subprograma** → Sentencias de Python con una misma sangría (4 espacios es el estándar) respecto al margen izquierdo, donde comienza el encabezado.

Los nombres de las subrutinas deben resumir la tarea que realizan, estos nombres son identificadores por tanto deben seguir una serie de reglas y no deben coincidir con ninguna palabra reservada del lenguaje. Es muy importante la **indentación**, (ya que en Python no existen llaves para separar el código que pertenece a una función) utilizaremos el **sangrado de 4 espacios** respecto al margen izquierdo. El final de la función se notará con la palabra **return** seguida del valor que devuelve, aunque no es obligatorio ya que depende del tipo de subrutina.

Tipos de subrutinas:

1. **Funciones** → Devuelven un solo valor. Al final de la función nos encontraremos con la palabra reservada **return** seguida del valor que devuelve.
2. **Procedimientos** → Pueden devolver ninguno ó más de un valor mediante **return**.

Para utilizar cualquier función que hayamos definido previamente tenemos que **llamar, invocar, o activar** la función. Las llamadas a función, se realizan en las instrucciones que forman parte del cuerpo principal del programa. Contienen el nombre de la función seguido de la lista de parámetros entre paréntesis, en caso de no tener lista de parámetros los paréntesis siguen siendo necesarios, si la función devuelve algún ó algunos valores mediante **return**, esos valores habrá que asignárselos a variables del mismo tipo.

1.2. FLUJO DE EJECUCIÓN.

Para poder entender un programa correctamente hay que saber el orden en el que las sentencias se ejecutan, este orden de ejecución se denomina **flujo de ejecución**.

La ejecución comienza con la primera sentencia del cuerpo principal del programa. Las sentencias se ejecutan una a una.

Las definiciones de las funciones no alteran el flujo de ejecución del programa, pero las llamadas a función sí. En este caso, cuando llamamos a una función estamos haciendo un desvío en el flujo de ejecución. En vez de continuar con la siguiente sentencia, el flujo salta a la primera línea de la función llamada, la ejecuta de arriba a abajo, y cuando acaba vuelve a la línea donde se encontraba previamente.

Se puede definir una función dentro de otra. En este caso la definición interna no se ejecuta hasta que la función externa no sea llamada. También podemos llamar a una función dentro de otra. En este caso tendremos que distinguir entre los diferentes tipos de variables que existen dependiendo de dónde se le asigne el valor y dónde se utilicen.

1.3. VARIABLES.

La principal ventaja de las funciones es que podemos reutilizar el código. Pero tenemos que tener cuidado con el nombre de las variables que utilizamos, ya que si utilizamos una función que contiene una variable auxiliar con el mismo nombre que una variable que ya utilizemos en el programa principal, los cambios que se produzcan en la variable dentro de la función podrían afectar de manera imprevista al programa principal. Por tanto, hay que saber diferenciar los tipos de variables según el lugar que ocupan en el programa para no cometer errores.

Se dispone de 3 tipos de variables:

- **Variables locales** → Son las variables sólo existen dentro de la función.
- **Variables globales** → Son las variables que forman parte del programa principal.
- **Variables no locales** → Son las variables que pertenecen a un ámbito superior al de la subrutina pero que no son globales.

En los programas que solamente contienen **funciones** que no contienen a su vez funciones, todas las variables libres son variables **globales**. En cambio, si el programa contiene una función que a su vez contiene otra función, las variables de las **subrutinas** pueden ser **globales** (si pertenecen al programa principal) o **no locales** (si pertenecen a la función).

Para identificar las variables libres existen las palabras reservadas **global** y **nonlocal**. Las variables locales no necesitan identificación.

A continuación unos ejemplos de cada tipo de variable:

- **Variables locales:**

```
# Archivo: ejemplo1.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Variables locales.

# Definición de función
def subrutina():
    a = 2
    print(a)

# Programa principal
a = 5
subrutina()
print(a)
```

Este programa contiene una función y dos valores distintos para la variable “a” dentro de la función vale 2 y fuera vale 5. El flujo de ejecución es el siguiente:

- 1) a = 5. En el programa principal.
- 2) Llamo a la función subrutina.
- 3) a = 2. Dentro de la función.
- 4) Imprimo el valor de 2 (a = 2). **Variable local.**
- 5) Vuelvo al programa principal y ejecuto la ultima instrucción que imprime el valor de 5 (a = 5). Valor de la variable global diferente a la local definida en la función
- 6) Fin del programa.

```
# Archivo: ejemplo2.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Variable local no declarada ni fuera ni dentro de la función, error.

# Definición de función
def subrutina():
    a = 2
    print(a)

# Programa principal
subrutina()
print(a)
```

Este programa contiene una función en la que se asigna un valor a la variable “a”. El problema es que no se la ha asignado ningún valor en el programa principal, por tanto

no funciona, ya que la variable **a** es **local** y su valor se destruye al acabar la función. Además el interprete busca en primer lugar el valor de la variable global **a** y como no esta definida y no puede imprimir ningún valor y devuelve un **error**.

```
# Archivo: ejemplo3.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Variable local que se le asignan dos valores
fuera y dentro de la función por tanto da error.

# Definición de función
def subrutina():
    print(a)
    a = 2
    print(a)
    return

# Programa principal
subrutina()
a = 5

print(a)
```

Este programa contiene una función y dos valores distintos para la variable “a” fuera de la función vale 5 (variable global) y dentro de la función se ha definido otra variable que aunque tiene el mismo nombre es diferente, su valor es 2. Este programa devuelve un **error** porque la asignación **a = 2** es **posterior a la impresión del valor**

- **Variables libres (globales y no locales):**

Si dentro de una función **no se le asigna un valor** a una variable, Python la considera **libre**, y busca un valor en niveles superiores. Si le encuentra algún valor en un nivel intermedio la variable es **no local**, y si se le encuentra un valor en el programa principal la variable es **global**.

A continuación unos ejemplos de variables libres:

```
# Archivo: ejemplo4.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Variable global sin declarar.

# Definición de función
def subrutina():
    print(a) #ESTO EN ESTRUCTURADA ESTA PROHIBIDO USO DE VAR
GLOBAL
```

```

    return

# Programa principal
a = 5
subrutina()
print(a)

```

Este programa contiene una función y un único valor para la variable “a” en el programa principal. El flujo de ejecución es el siguiente:

- 1) a = 5.
- 2) Llamo a la función subrutina.
- 3) La variable “a” es libre, por lo tanto Python busca en niveles superiores su valor, en este caso es en el **programa principal**. Imprimo el valor **5** (a = 5).
Variable global.
- 4) Vuelvo al programa principal y ejecuto la ultima instrucción que imprime el valor de **5** (a = 5).
- 5) Fin del programa.

```

# Archivo: ejemplo5.py
# Autor: XXXXX XXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Variable no local sin declarar.

# Definición de funciones
def subrutina():
    def sub_subrutina():
        print(a)

    a = 2
    sub_subrutina()
    print(a)

# Programa principal
a = 5
subrutina()
print(a)

```

Este programa contiene una función y dos valores distintos para la variable “a” dentro de la función vale 2 y fuera vale 5. El flujo de ejecución es el siguiente:

- 1) a = 5. En el programa principal.
- 2) Llamo a la función subrutina.
- 3) Defino la función sub_subrutina.
- 4) a = 2. Dentro de la función.
- 5) Llamo a la función sub_subrutina.

- 6) La variable “a” es libre, por lo tanto Python busca en niveles superiores su valor, en este caso es en la **función**. Imprimo el valor de **2** (a = 2). **Variable no local**.
- 7) Vuelvo a la función y ejecuto la ultima instrucción que imprime el valor de **2** (a = 2).
- 8) Vuelvo al programa principal y ejecuto la última instrucción que imprime el valor de **5** (a = 5).
- 9) Fin del programa.

```
# Archivo: ejemplo6.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Variable libre no declarada a la que no se le
# asigna ningún valor por lo tanto da error.

# Definición de función
def subrutina():
    print(a)

# Programa principal
subrutina()
print(a)
```

En este programa la variable “a” se considera **libre**, por lo tanto Python busca en niveles superiores su valor, al no encontrarlo tampoco en el programa principal da un **error**.

- **Variables declaradas (global y nonlocal):**

Si dentro de una función **no se le asigna un valor** a una variable, Python la considera **libre**, y busca un valor en niveles superiores. Si le encuentra algún valor en un nivel intermedio la variable es **no local**, y si se le encuentra un valor en el programa principal la variable es **global**.

NOTA: En esta asignatura no usaremos JAMÁS variables globales o no locales dentro de las funciones, porque lo que nos interesa de las funciones es poder reutilizarlas en otros programas, y al declarar variables no locales o globales dentro nos impide reutilizarlo. Nosotros estamos aprendiendo la programación de lenguajes estructurados con la sintaxis de Python que es más sencilla, Python es un lenguaje orientado a objetos.

```
# Archivo: ejemplo7.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Variable global declarada dentro de una
función.
```

```

# Definición de función
def subrutina():
    global a# NO HACER ESTO EN ESTRUCTURADA
    print(a)
    a = 2

# Programa principal
a = 5
subrutina()
print(a)

```

Este programa contiene una función en la que la variable “a” se declara **global**, por lo tanto la variable “a” del programa y de la función es la misma. El flujo de ejecución es el siguiente:

- 1) a = 5. En el programa principal.
- 2) Llamo a la función subrutina.
- 3) Declaro la variable “a” como variable global.
- 4) Imprimo el valor **5** (a = 5). Al declararla como variable global utilizo el valor 5 que se le asigna en el programa principal.
- 5) Cambio el valor de la variable “a”. **a = 2**.
- 6) Vuelvo al programa principal y ejecuto la última instrucción que imprime el valor de **2** (a = 2). Ya que al ser la misma variable dentro de la función modifiqué su valor.
- 7) Fin del programa.

```

# Archivo: ejemplo8.py
# Autor: XXXXX XXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Variable no local declarada dentro de una
función

# Definición de funciones
def subrutina():
    def sub_subrutina():
        nonlocal a
        print(a)
        a = 1

    a = 2
    sub_subrutina()
    print(a)

```

```
# Programa principal
a = 5
subrutina()
print(a)
```

Este programa contiene una función en la que la variable “a” se declara **nonlocal**, por lo tanto la variable “a” de la subfunción y de la función es la misma. El flujo de ejecución es el siguiente:

- 1) a = 5. En el programa principal.
- 2) Llamo a la función subrutina.
- 3) Defino la función sub_subrutina.
- 4) a = 2. En la función.
- 5) Llamo a la función sub_subrutina.
- 6) Declaro la variable “a” como **variable no local**.
- 7) Imprimo el valor 2 (a = 2). Al declararla como variable no local utilizo el valor 2 que se le asigna en la función.
- 8) Cambio el valor de la variable “a”. a = 1.
- 9) Vuelvo a la función y ejecuto la última instrucción que imprime el valor de 1 (a = 1). Ya que al ser la misma variable en las dos funciones se ha modificado su valor.
- 10) Vuelvo al programa principal y ejecuto la última instrucción que imprime el valor de 5 (a = 5). Valor que no ha sido modificado.
- 11) Fin del programa.

```
# Archivo: ejemplo9.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Variable declarada no local que encuentra su
valor
# en el programa principal por lo tanto da error porque no
puede
# ser local y global a la vez.

# Definición de funciones
def subrutina():
    def sub_subrutina():
        nonlocal a
        print(a)
        a = 1

    sub_subrutina()
    print(a)

# Programa principal
```

```
a = 5
subrutina()
print(a)
```

Este programa contiene una función en la que la variable “a” se declara **nonlocal**, por lo tanto la variable “a” de la subfunción y de la función es la misma. Por otro lado la variable “a” de la función al no tener asignado un valor se considera global, ya que en el nivel superior (programa principal) se le asigna un valor. Existe un error de sintáxis porque una variable no puede ser no local y global a la vez, por lo tanto da **error**.

1.4. PARÁMETROS Y DEVOLUCIÓN DE VALORES.

Los parámetros o argumentos de una función constituyen el modo de pasar información del programa principal a la función y viceversa. Podemos tener varios parámetros tanto de entrada como de salida o ninguno.

Vamos a ver varios ejemplos de paso de parámetros en subrutinas:

- **Funciones sin parámetros.**

Para definir e invocar funciones sin parámetros son obligatorios los paréntesis a continuación del identificador.

Ejemplo: Realizar una función que lee de teclado una cadena de caracteres.

```
# Archivo: ejemplo10.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Realizar una función que lee de
# teclado una
# cadena de caracteres.

# Definición de función
def lee_cadena():
    return str(input())

# Programa principal
a = lee_cadena()
print(a)
```

Este programa contiene la función lee_cadena() que lo único que hace es devolver una cadena leída por teclado, y al imprimir la variable “a” vemos la cadena que ha leído.

Nota: NORMALMENTE EN PROGRAMACIÓN ESTRUCTURADA LAS SUBROUTINAS LLEVAN PARÁMETROS

- **Funciones con un parámetro de entrada.**

Para definir e invocar funciones con un parámetro es obligatorio que el parámetro se encuentre entre paréntesis a continuación del identificador.

Ejemplo: Realizar una función que dada una edad diga si es mayor o menor de edad.

```
# Archivo: ejemplo11.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Realizar una función que dada una
edad diga
# si es mayor o menor de edad.

# Definición de función
def mayor_edad(edad):
    if edad < 18:
        a = False
    else:
        a = True
    return a

# Programa principal
n = int(input('Dime edad: '))
b = mayor_edad(n)
print (b)
```

- **Funciones con varios parámetros de entrada.**

Para definir e invocar funciones con varios parámetros es obligatorio que los parámetros se encuentren entre paréntesis y separados por comas, a continuación del identificador. Si queremos tener un número indefinido de parámetros dentro de los paréntesis de la función tenemos que escribir → ***args**

Ejemplo: Realizar una función que calcule el área de un rectángulo dada la altura y el ancho.

```
# Archivo: ejemplo12.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Realizar una función que de el área
de un
# rectángulo dadas la altura y el ancho.

# Definición de funciones
def area_rectangulo(altura, ancho):
    área = altura*ancho
    return área
```

```
# Programa principal
print (area_rectangulo(5, 9))
```

Hasta ahora todos los ejemplos que hemos realizado son funciones por tanto dado unos parámetros de entrada devuelven un valor.

NOTA: TODOS LOS PARÁMETROS QUE HEMOS UTILIZADOS SON POR VALOR, (copia de valor).

- **PROCEDIMIENTOS.**

Los procedimientos son subprogramas que **devuelven ninguno o varios valores**. Muchas veces sirven para mostrar mensajes por pantalla, pero eso no es devolver ningún valor, no hay que confundirlo.

Ejemplo: Realizar una función que imprima la media de 2 numeros introducidos por teclado.

NOTA: En vez de definir la función con dos parámetros, se ha puesto ***args** que significa que puedo utilizar esa función con el número de parámetros que quiera.

```
# Archivo: ejemplo13.py
# Autor: XXXXX XXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Función que imprima la media de 2
números introducidos por teclado.

# Definición de función
def escribe_media(*args):
    total = 0
    for i in args:
        total += i
    resultado = total / len(args)
    print (resultado)

# Programa principal
a = int(input('Escribe un numero: '))
b = int(input('Escribe otro numero: '))
escribe_media(a, b)
```

NOTA: En los lenguajes estructurados las subrutinas que **devuelven 1 sólo valor** se llaman **funciones**, y los que **devuelven más de uno o ninguno** se llaman **procedimientos**.

1.5. OTROS TIPOS DE DATOS.

Los valores que asignamos a las variables (números, cadenas, tuplas, listas, etc) se almacenan en **objetos**. Las variables son simples etiquetas para hacer referencia a los objetos, pero son independientes de ellos.

Tenemos dos tipos de objetos:

- **Objeto inmutable** → Son objetos que no se pueden modificar, como por ejemplo los números, cadenas o tuplas.
- **Objeto mutable** → Son objetos que se pueden modificar, como por ejemplo las listas y los diccionarios.

Si modificamos una variable que hace referencia a un objeto inmutable, la variable pasa a hacer referencia a otro objeto. En cambio en los objetos mutables podemos modificar la variable o el objeto al que hace referencia.

El paso de los argumentos a una función en la mayoría de lenguajes se puede hacer de dos formas:

- **Paso por valor** → Se envía el valor de la variable, en cuyo caso la función no puede modificar la variable, conoce su valor, pero no el objeto que lo almacena.
- **Paso por referencia** → Se envía la dirección de memoria de la variable, en cuyo caso la función sí puede modificar la variable

NOTA: En Python no es así, cuando se pasa una variable como argumento en una función **se envía la referencia al objeto al que hace referencia la variable**. Si el objeto es mutable podrá modificar la función, en cambio si el objeto es inmutable no podrá hacerlo (Nosotros en este curso obviamos esto y solamente simulamos los parámetros por valor y por referencia).

Para simular el paso de argumentos como en los lenguajes estructurados con la sintaxis de Python se hace de la siguiente forma:

- **Paso por valor** → El argumento que se pasa por valor únicamente aparece en la cabecera de la función pero no en el **return**, **esto hace que dicho argumento no modifique la variable global**.

Ejemplo:

```
Def función (x)
...
    Return y
```

El parámetro `x` es un parámetro pasado por valor, ya que únicamente se encuentra en la cabecera de la función, no es retornado al programa principal con la sentencia `return`.

- **Paso por referencia** → El argumento que se pasa por referencia aparece tanto en la cabecera de la función como en el `return`, luego hay que asignárselo en el cuerpo principal a la variable global para que esta modifique su valor.

Ejemplo:

```
Def función (x,z)
...
    Return x # x es una variable de E/S PASO POR
              REFERENCIA. z variable de E PASO POR
              VALOR
```

```
Def función (x,z)
...
    y =x+z
    Return y # Y es una variable de S, PASO POR
              REFERENCIA. x, z variables de E PASO
              POR VALOR
```

CURIOSIDADES DE PYTHON: (Apartado que muestra realmente como funciona Python, nosotros solamente nos centraremos en los pasos de parámetros por valor y por referencia) A continuación explicaremos ejemplos de objetos mutables e inmutables, aunque entraremos en más detalles en el tema siguiente en el que se ven listas, tuplas, diccionares, etc...

Ejemplo objeto inmutable:

```
def aumenta(x):
    print(id(x))
    x += 1
    print(id(x))
    return x

a = 3
print(id(3), id(4))
print(id(a))
print(aumenta(a))
print(a)
```

SALIDA POR PANTALLA:

```
1569378800 1569378816
1569378800
1569378800
1569378816
4
3
1569378800
```

```
print(id(a))
```

La variable que se pasa como argumento a la función es un número entero, por lo tanto un objeto inmutable. Al intentar modificar esta variable dentro de la función, Python lo que hace es crear un objeto nuevo para guardar la nueva variable que aumenta dentro de la función. Se observa que los identificadores cambian, ya que se ha creado un nuevo objeto.

Ejemplo objeto mutable:

```
def aumenta(x):
    print(id(x))
    x += [1]
    print(id(x))
    return x

a = [3]
print(id(a))
print(aumenta(a))
print(a)
print(id(a))
```

SALIDA POR PANTALLA:

```
48317784
48317784
48317784
[3, 1]
[3, 1]
48317784
```

En este caso la variable que se pasa como argumento a la función es una lista, por lo tanto un objeto mutable, es decir, dentro de la función se puede modificar su valor. Se observa que el identificador es siempre el mismo ya que dentro de la función lo que se hace es aumentar la variable, pero no crear un objeto nuevo como en el caso anterior.

NOTA: En Python tenemos objetos, pero como estamos simulando la programación estructurada, el concepto de OBJETO no se utiliza, por lo tanto en la práctica no se utilizará, ya que la programación que estamos aprendiendo es una programación estructurada con la sintaxis de Python.

FUNCIONES INTERNAS DE PYTHON.

Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Listado de funciones de Python 3.4.

En la página web oficial de Python se puede encontrar el listado de funciones internas dependiendo de la versión que tengamos instalada.

1.6. EJERCICIOS.

1.- Diseñar una función que permita obtener el valor absoluto de un número.

```
# Archivo: ejercicio1.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Diseñar una función que permita obtener el
valor
# absoluto de un número.

# Definición de función
def absoluto(x):
    if x >= 0:
        numero = x
    else:
        numero = -x

    return numero

# Programa principal
a = int(input('Escribe un número: '))
b = absoluto(a)
```

```
print(' El valor absoluto es',b) #Paso de parámetro de a a
x por valor, el valor de a se copia en x pero no se
modifica
```

2.- Realizar un procedimiento que obtenga la división entera y el resto de la misma utilizando únicamente los operadores suma y resta.

Se Supone que dividendo es mayor o igual a divisor.

```
# Archivo: ejercicio2.py
# Autor: XXXXX XXXXXXXX (nombre y apellidos)
# Fecha: DD/MM/AAAA
# Descripción: Realizar un procedimiento que obtenga la
#división entera y el resto de la misma utilizando los
#operadores suma y resta.

# Definición de función
def division(dividendo,divisor):
    cociente = 0
    while (dividendo >= divisor):
        dividendo = dividendo - divisor
        cociente = cociente + 1

    resto = dividendo

    return cociente, resto

# Programa principal
x = int(input('Introduzca dividendo: '))
y = int(input('Introduzca divisor: '))

cociente, resto = division(x,y)
print('El cociente es', cociente, 'y el resto es', resto)
# x, y son parámetros por valor es decir datos de entrada
se copia #el valor en dividendo y divisor.

#cociente y resto paso de parámetro por referencia entran
sin nada y al salir del procedimiento tienen el valor de
cociente y resto, por tanto se modifica el valor que
contenían.
```

ENUNCIADOS EJERCICIOS



TEMA 2: FUNCIONES Y PROCEDIMIENTOS

TEMA2:

ENUNCIADOS

1. Diseñar un programa para el cálculo de la expresión calcule x^y
2. Realizar un programa que pregunte al usuario si quiere continuar ejecutándose
3. Escribir un programa que compruebe si un carácter dado es o no un dígito
4. Escribir un programa que contenga una función que recibe una base distinta de diez y un números en base decimal y escribe en pantalla el resultado de cambiar el número dado a la nueva base
5. Realizar un programa para obtener el máximo y mínimo de una serie de números enteros solicitados al usuario
6. Realizar un programa que decida si son o no primos una serie de números solicitados al usuario
7. Realizar un programa que lea un número arbitrario de enteros y cuente el número de valores positivos y números que se han introducido.
8. Programa que comprueba si un carácter dado es una mayúscula del alfabeto inglés
9. Programa que comprueba si un carácter dado es una mayúscula del alfabeto español
10. Escribir un programa que pida las tres componentes numéricas de una fecha y devuelva esa fecha y la fecha nueve meses posterior, escribiendo en ambos casos el mes mediante un nombre (enero, febrero, etc.). Utilizar subprogramas.
11. Un funicular viaja por un cable de 1000 metros que está tendido entre dos torres con una torre intermedia. Su velocidad depende de la distancia a la torre más cercana. Así, si se encuentra a menos de 30 metros de una torre la velocidad en metros por segundo es: $V=2.425 + 0.00175D^2$ y en caso contrario es $V=0.625 + 0.12D - 0.00025D^2$, siendo D la distancia a la torre más cercana en metros. Escribir una función booleana que devuelva verdadero si el funicular está a menos de 30 metros de una torre, una función que devuelva la distancia a la torre más cercana y otra que calcule la velocidad del funicular. Con estas funciones componer un programa que pida la distancia a la torre de la izquierda y calcule la velocidad del funicular, indicando además la distancia a la torre más cercana.
12. Realizar un programa que dibuje con * un rectángulo dada su base y altura ó un triángulo de una altura determinada girado 90° (los datos para crear las figuras deberán ser solicitados por el usuario)

Ejemplo:

Opción 1 RECTÁNGULO

Base =2

Altura = 4

**
**
**
**

Opción 2 TRIÁNGULO

Altura = 4

*
**

**
*