FUNCIONES

Fundamentos de Informática

Grado en Ingeniería Química

Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación

UC | ETS de Ingenieros Industriales y de Telecomunicación

Diseño descendente

En programación es importante elegir el diseño adecuado a cada problema.

Una técnica muy utilizada es la del diseño descendente, *top-down*, que consiste en dividir el problema en subproblemas, que se pueden tratar y codificar de forma separada.

En Python, hay dos tipos de subprogramas:

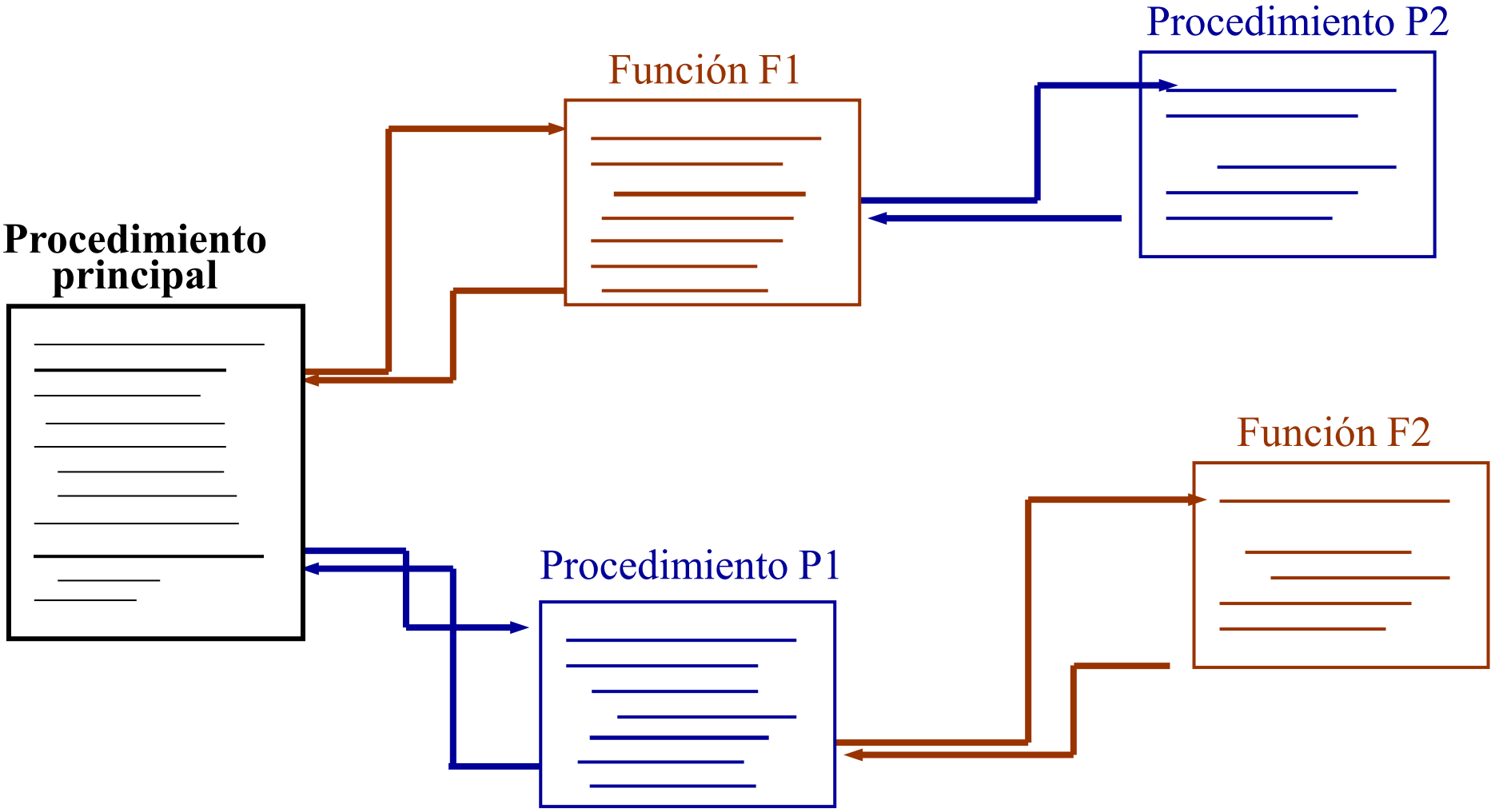
- ***Funciones***
- ***Procedimientos***

Inicialmente, cada subprograma se puede ejecutar en la Shell de Python hasta tenerlo a punto.

Al final, se combina todo para construir el programa final. Generalmente hay un procedimiento principal y otro(s) subprograma(s) que son invocados desde el principal cuando es necesario. Cuanto más se invoque a un subprograma, más sentido tiene haberlo codificado por separado.

Además, un subprograma se puede usar en muchos programas.

Programación con funciones y procedimientos



Ventajas del diseño descendente (top-down)

Es mucho más fácil encontrar errores en el código, sobre todo en programas largos.

Permite usar procedimientos y funciones construidos por otros programadores.

Evita cambios indeseables en las variables del programa. Sólo algunas de ellas se transfieren como argumentos entre las diferentes unidades de programa, aquellas que son necesarias para realizar los cálculos previstos, y de ellas solo las listas y diccionarios pueden modificarse.

Las demás variables sólo son accesibles en la unidad de programa donde se usan **(variables locales)**, quedando a salvo de cambios imprevistos para el resto de las unidades de programa.

Funciones

Hay dos tipos de funciones:

Predefinidas: las que suministra el propio lenguaje que pueden usarse directamente o importando previamente los módulos o librerías donde se definen.

Definidas por el programador: que permiten responder a necesidades particulares del usuario, no proporcionadas por las funciones predefinidas.

Las funciones definidas por el programador se usan igual que las funciones predefinidas, pueden formar parte de expresiones y aparecer en todos aquellos lugares donde se puede usar una expresión.

Su resultado (respuesta) es un valor numérico, lógico, cadena, lista, tupla, diccionario...

La estructura general de una función es:

- Cabecera de la función
- Cuerpo de la función

Funciones (definición)

```
def nombre_función ([Lista de argumentos formales]):
        ["" documentación de la función ""]
        [bloque]
        return  resultado
```

Cabecera

Cuerpo de la función

Funciones (definición)

Cabecera de la función:

def nombre_función **(**[Lista de argumentos formales]**):**

Es la primera sentencia no comentada de la función e identifica esa unidad de programa como una función.

nombre_función es cualquier identificador válido de Python.

Lista de argumentos formales es una secuencia (puede ser vacía) de constantes, variables o expresiones, separadas por comas. Se emplean para pasar información al cuerpo de la función.

Cuerpo de la función:

Es aconsejable, pero no obligatorio, empezar con una sentencia de documentación, embebida entre tres comillas simples o dobles, donde se resume la tarea que realiza la función, qué recibe como entrada (y de qué tipo es) y qué devuelve como resultado (y de qué tipo es).

Debe incluir al menos una sentencia **return** con el resultado a transferir. Al ejecutarse se devuelve el control a la línea de la unidad de programa donde se hizo la llamada o invocación a la función.

Funciones (invocación)

Una función se invoca escribiendo:

```
nombre_función ([Lista de argumentos reales])
```

formando parte de una expresión en cualquier lugar donde puede aparecer una expresión.

Al evaluar la función en sus argumentos reales se devuelve un valor que es usado para evaluar, a su vez, la expresión de la que forme parte esa función.

El número, tipo y orden de los argumentos reales que aparecen en la llamada a la función y los argumentos formales que aparecen en la cabecera de su definición deben corresponderse.

Ejecución de la llamada a una función

- **Definición:**

  def nombre_función **(**[lista de argumentos formales]**)**

- **Invocación:**

  nombre_función ([lista de argumentos reales])

1. Se evalúan los argumentos reales que son expresiones.
2. Se *asocian* los argumentos reales con sus correspondientes formales.
3. Se ejecuta el cuerpo de la función especificada.
4. Cuando se alcanza la sentencia **return** se devuelve el control a la unidad de programa que hizo la llamada, en concreto, a la sentencia donde se invocó a la función, sustituyendo su nombre por el resultado devuelto por la función.

Transferencias por valor o dirección

En Python, la transferencia de argumentos entre dos unidades de programa se realiza:

- **Por valor** si el tipo de variable es **int, float, str, tuple** y

- **Por dirección** si el tipo de variable es **list, dict**.

Por valor: se crea una copia local de la variable dentro de la función. Los enteros, reales, cadenas y tuplas son argumentos de entrada.

Por dirección: se maneja directamente la lista o diccionario dentro de la función. Por tanto, si se realizan cambios dentro de la función a esa lista o diccionario le afectarán también fuera de esa función. En ese caso, las listas y diccionarios son argumentos de entrada y salida.

Función sin argumentos. Ejercicio 1a

Construye una función para escribir en pantalla un menú con opciones y devolver la opción elegida. Por ejemplo, una calculadora rudimentaria que sume, reste, multiplique o divida.

cap5_1a.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_1a.py (3.10.7)

File Edit Format Run Options Window Help

```
1  #PB Nov 2022
2  def menu():
3      '''presenta un menú en pantalla y devuelve la opción elegida'''
4      while True:
5          print("Pulse 1 para sumar")
6          print("Pulse 2 para restar")
7          print("Pulse 3 para multiplicar")
8          print("Pulse 4 para dividir")
9          print("Pulse 5 para salir")
10         opcion=int(input())
11         if 1<=opcion<=5: break
12         print("Opción incorrecta")
13         print("Inténtelo de nuevo")
14     return opcion
```

Ejecutando el archivo de arriba en el entorno IDLE: Run -> Run Module se define la función anterior, se aprende qué hay que hacer. Para usarla o invocarla escribir, por ejemplo, en la Shell:

>>>menu()

Función sin argumentos. Ejercicio 1b

Construye un programa que invoque a la función anterior y solicite dos números por teclado simulando una calculadora rudimentaria.

cap5_1b.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_1b.py (3.10.7)

File Edit Format Run Options Window Help

```
#PB Nov 2022
def menu(): #función
    '''presenta un menu en pantalla y devuelve la opción elegida'''
    opcion=0
    while opcion<1 or opcion>5:
        print("Pulse 1 para sumar")
        print("Pulse 2 para restar")
        print("Pulse 3 para multiplicar")
        print("Pulse 4 para dividir")
        print("Pulse 5 para salir")
        opcion=int(input())
    return opcion
op=menu()
while op!=5:
    a=float(input("Dame un número "))
    b=float(input("Dame otro número "))
    if op==1:
        print(a,"+",b,'=',a+b)
    elif op==2:
        print(a,"-",b,'=',a-b)
    elif op==3:
        print(a,"*",b,'=',a*b)
    elif op==4 and b!=0:
        print(a,"/",b,'=',a/b)
    elif op==4 and b==0:
        print("No se puede dividir entre cero")
    op=menu()
print("Gracias por usar este programa")
```

Invocando, al volver se sustituye menu() por la opción devuelta.

Invocando, al volver se sustituye menu() por la opción devuelta

Procedimientos

Son funciones que no devuelven nada.

Un procedimiento especial es el procedimiento principal, generalmente dedicado a leer los datos de entrada (bien por teclado, como argumentos de entrada al procedimiento o en un archivo de entrada), realizar las llamadas a las demás funciones y procedimientos y escribir los resultados (bien en pantalla o en un archivo de salida).

Ejemplo: construye una calculadora rudimentaria.

- procedimiento principal()
- función menú()

Un procedimiento especial: el principal. Ejercicio 1c

cap5_1c.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_1c.py (3.10.7)

File Edit Format Run Options Window Help

```
#PB Nov 2022
def principal():    #procedimiento
    '''solicita dos números reaales por teclado y opera con ellos'''
    op=menu()
    while op!=5:
        a=float(input("Dame un número "))
        b=float(input("Dame otro número "))
        if op==1:
            print(a,"+",b,'=',a+b)
        elif op==2:
            print(a,"-",b,'=',a-b)
        elif op==3:
            print(a,"*",b,'=',a*b)
        elif op==4 and b!=0:
            print(a,"/",b,'=',a/b)
        elif op==4 and b==0:
            print("No se puede dividir entre cero")
        op=menu()
    print("Gracias por usar este programa")

def menu(): #función
    '''presenta un menu en pantalla y devuelve la opción elegida'''
    opcion=0
    while opcion<1 or opcion>5:
        print("Pulse 1 para sumar")
        print("Pulse 2 para restar")
        print("Pulse 3 para multiplicar")
        print("Pulse 4 para dividir")
        print("Pulse 5 para salir")
        opcion=int(input())
    return opcion
#principal()
```

Diferencias entre los Ejercicios 1b y 1c I

EJECUTAR

- Ejercicio 1b en IDLE: Run -> Run Module define la función menu() y ejecuta el programa normalmente. No se puede escribir la función menu() debajo del programa, pues este se interpreta de arriba abajo. La primera sentencia en la que se invoca a menu() daría error, pues la función es aún desconocida.
- Ejercicio 1c en IDLE: Run -> Run Module define el procedimiento principal() y la función menu(). Da igual el orden de los dos subprogramas en el archivo, ambos van a quedar definidos. Para ejecutar el programa, invocar al procedimiento principal(), por ejemplo, desde la Shell:

  >>>principal()

Diferencias entre los Ejercicios 1b y 1c II

VARIABLES LOCALES Y GLOBALES

- Ejercicio 1b. Las variables fuera de la función menu() son globales. Por tanto, automáticamente son accesibles y modificables (solo para los tipos list y dict) en la función.
- Ejercicio 1c. Las variables en la función menú() son locales a esa función y lo mismo ocurre con las variables del procedimiento principal(). Solo tienen sentido en el interior de la función /procedimiento que las define.

CONCLUSIÓN

Se recomienda compartimentar todo el programa en funciones y /o procedimientos para evitar cambios indeseables en listas y diccionarios.

Función con un argumento. Ejercicio 2a

Calcular un número combinatorio sabiendo que m debe ser mayor o igual que n. $\binom{m}{n} = \frac{m!}{n!(m-n)!}$.

a. Usar la función factorial() de la librería math.

cap5_2a.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_2a.py (3.10.7)

File Edit Format Run Options Window Help

```
1 #PB Nov 2022
2 from math import factorial
3 m=int(input("m "))
4 n=int(input("n "))
5 if m>=n:
6     resultado=factorial(m)/(factorial(n)*factorial(m-n))
7     print(m,' sobre ',n, 'es ',resultado)
8 else:
9     print("m<n")
```

Función con un argumento. Ejercicio 2b I

b. Crear una función personal para calcular un factorial.

cap5_2b.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_2b.py (3.10.7)

File Edit Format Run Options Window Help

```
1  #PB Nov 2022
2  def fact(num):
3      '''Calcular el factorial de un entero'''
4      acum=1
5      for i in range(num,1,-1):
6          acum*=i
7      return acum
8  def main():
9      m=int(input("m "))
10     n=int(input("n "))
11     if m>=n:
12         resultado=fact(m)/(fact(n)*fact(m-n))
13         print(m,' sobre ',n, 'es ',resultado)
14     else:
15         print("m<n")
```

Como el argumento de fact es de tipo int se envía el valor.

Función con un argumento. Ejercicio 2b II

Python 3.6
(known limitations)

```
1  def fact(num):
2      '''Calcular el factorial de un natural'''
3      acum=1
4      for i in range(num,1,-1):
5          acum*=i
6      return acum
7  def main():
8      m=int(input("m "))
9      n=int(input("n "))
10     if m>=n:
11         resultado=fact(m)/(fact(n)*fact(m-n))
12         print(m,' sobre ',n, 'es ',resultado)
13     else:
14         print("m<n")
15 main()
```

Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Print output (drag lower right corner to resize)

```
m 6
n 3
```

Frames Objects

Global frame
- fact → function fact(num)
- main → function main()

main
- m 6
- n 3

fact
- num 6
- acum 720
- i 2
- Return value 720

Transferencias por valor: int

¿Qué se escribe en pantalla?

```
1 def prueba(z):
2       z=7
3 def principal():
4       x=5
5       prueba(x)
6       print('x ',x)
```

Se escribe x 5

Los enteros **int** se envían a las funciones/procedimientos por **valor.**

Se hace una copia del valor 5 y se almacena en la variable z.

Aunque se modifique z a 7 en prueba, el cambio no repercute en la variable x de principal.

Es imposible modificar x dentro del procedimiento prueba, pues no se sabe donde está almacenada!!

x es una variable **local** al procedimiento principal, sólo está definida dentro de ese procedimiento y ocurre lo mismo con z en prueba.

Función con un argumento por defecto. Ejercicio 3

Crear una función para calcular $\sum_{i=1}^{n} \frac{1}{i!}$, tal que n es el argumento de entrada y vale 10 por defecto.

cap5_3.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_3.py (3.10.7)

File Edit Format Run Options Window Help

```
1 #PB Nov 2022
2 from math import factorial
3 def sumatorio(n=10):
4     suma=0
5     for i in range(1,n+1):
6         suma+=1/factorial(i)
7     return suma
```

>>>sumatorio()

>>>sumatorio(8)

Función con un argumento. Ejercicio 4

Crear una función para saber si un número es positivo. Se recibe un número y se devuelve un booleano. A continuación, construir un procedimiento que pida tres números por teclado e invoque a la función anterior escribiendo en pantalla los positivos.

cap5_4.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_4.py (3.10.7)

File Edit Format Run Options Window Help

```
1  #PB Nov 2022
2  def espositivo(x):
3      '''se recibe un número real y se devuelve un booleano'''
4      return x>0
5
6  def main():
7      for i in range(3):
8          num=float(input("Número "))
9          if espositivo(num):
10             print(num,'es positivo')
11 #main()
```

Notar cómo gana legibilidad la condición, por usar esta función.

Función con dos argumentos. Ejercicio 5

Crear una función para dividir dos números. Si el divisor es cero devolver un mensaje que lo indique.

cap5_5.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_5.py (3.10.7)

File Edit Format Run Options Window Help

```
1 #PB Nov 2022
2 def dividir(x,y):
3     '''dividir dos números y devolver el resultado'''
4     if y!=0:
5         return x/y
6     return 'El divisor es cero'
```

>>>dividir(7.5,10)

>>>dividir(8.7,0)

Transferencia de listas a funciones I. Ejercicio 6

Construir una función para calcular la media aritmética de una lista de números.

cap5_6.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_6.py (3.10.7)

File Edit Format Run Options Window Help

```
1 #PB Nov 2022
2 def fmedia(lista):
3     '''media aritmética de una lista'''
4     acum = 0
5     for elem in lista:
6         acum += elem
7     return acum/len(lista)
```

>>>fmedia([1.5,7.3,0.5,7.2,4.9])

Transferencia de listas a funciones II . Ejercicio 7

Construir una función para contar la cantidad de positivos, negativos y ceros de una matriz. Se recibe una lista de listas y se devuelve una tupla (positivos, negativos, ceros).

cap5_7.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_7.py (3.10.7)

File Edit Format Run Options Window Help

```
#PB Nov 2022
def cuenta_matriz(lista):
    '''contar positivos, negativos y ceros de una matriz
       se recibe una lista y se devuelve una tupla (posit,negat,ceros)'''
    p=0;n=0;c=0
    for fila in lista:
        for elem in fila:
            if elem>0:
                p+=1
            elif elem<0:
                 n+=1
            else:
                c+=1
    return p,n,c
```

```
>>>cuenta_matriz([[5,-1,7],[0,8,9]])
```

Función que invoca a otras dos. Ejercicio 8

Construir una función para calcular el área de un cilindro.

cap5_8.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_8.py (3.10.7)

File Edit Format Run Options Window Help

```
1 #PB Nov 2022
2 def areacilindro(r,h):
3     '''se reciben el radio y altura del cilindro'''
4     return 2*areacirculo(r)+arealateral(r,h)
5 from math import pi
6 def areacirculo(radio):
7     return pi*radio**2
8 def arealateral(radio,altura):
9     return 2*pi*radio*altura
```

pi puede usarse en cualquiera de las tres funciones. Es una constante notable "**global**". Su sentencia está <u>fuera</u> de las funciones.

>>>areacilindro(0.1,2.5)

Función que invoca a otras dos. Ejercicio 8

Python 3.6
(known limitations)

```
1  def areacilindro(r,h):
2      '''se reciben el radio y altura del cilindro'''
3      return 2*areacirculo(r)+arealateral(r,h)
4  from math import pi
5  def areacirculo(radio):
6      return pi*radio**2
7  def arealateral(radio,altura):
8      return 2*pi*radio*altura
9  areacilindro(0.1,2.5)
```

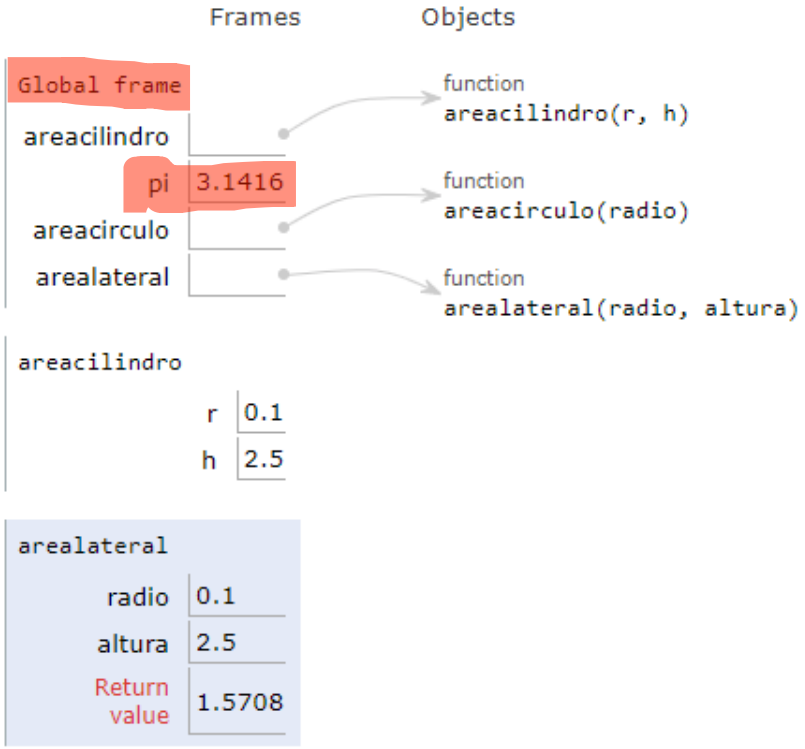
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 13 of 14

Customize visualization



Transferencias por dirección: listas I. Ejercicio 9

Construye dos procedimientos:

a. Un procedimiento desplaza_derecha que desplace los valores de las componentes de una lista, por ejemplo: A[5,10,15,20,25,30,35)] una posición hacia la derecha de modo que el valor de la última componente pase a la primera, es decir, después del desplazamiento, la lista es A[35,5,10,15,20,25,30].

b. Un procedimiento principal que reciba una lista y cuántos desplazamientos e invoque al procedimiento anterior desplaza_derecha tantas veces como desplazamientos quieran hacerse escribiendo la lista resultante en pantalla.

Transferencias por dirección: listas II. Ejercicio 9

a

b

cap5_9.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_9.py (3.10.7)

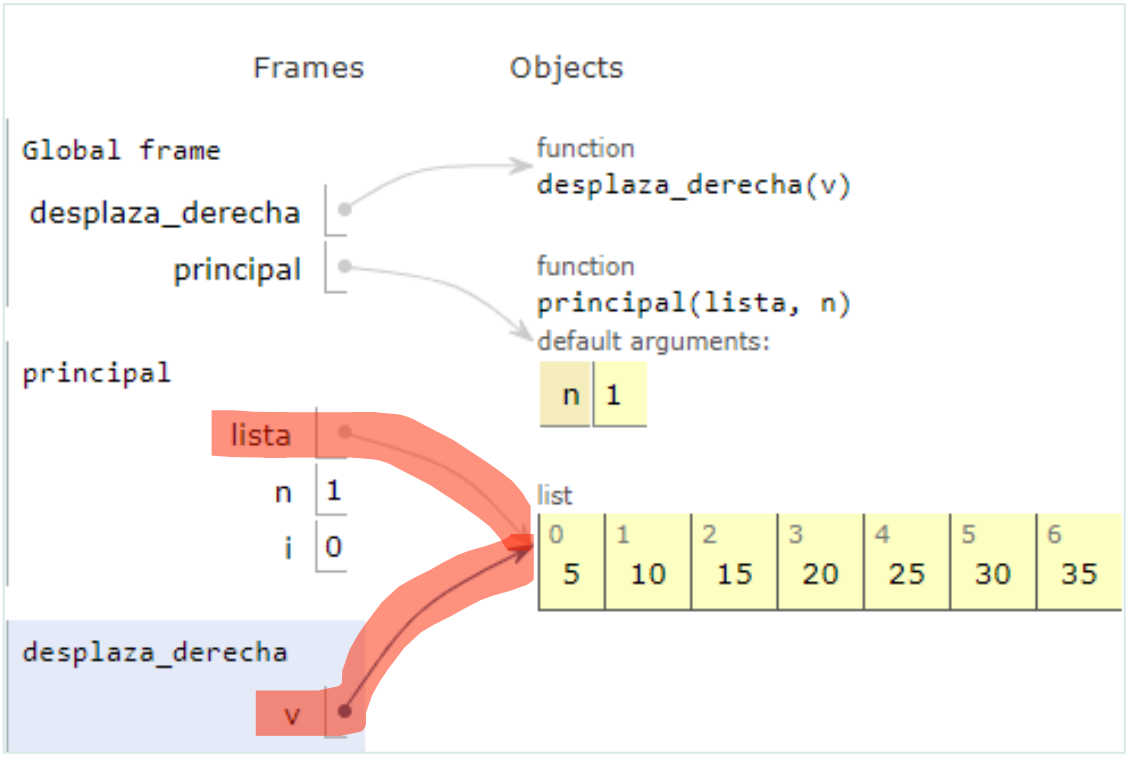
File Edit Format Run Options Window Help

```
1  #PB Nov 2022
2  #Procedimiento desplaza a la derecha
3  def desplaza_derecha(v):
4      aux=v[len(v)-1]
5      for i in range(len(v)-2,-1,-1):
6          v[i+1]=v[i]
7      v[0]=aux
8
9  #Contiene la lista y muestra ambas
10 def principal(lista,n=1):
11     for i in range(n):
12         desplaza_derecha(lista)
13     print('Lista desplazada derecha ',n,'veces',lista)
```

>>>principal([5,10,15,20,25,30,35],3)
Lista desplazada derecha 3 veces [25, 30, 35, 5, 10, 15, 20]

La lista se envía del principal a desplaza_derecha por **dirección**. Por tanto, las modificaciones de la lista en desplaza_derecha son visibles también en principal.

Transferencias por dirección: listas III. Ejercicio 9



v y lista son dos identificadores del mismo objeto, v en desplaza_derecha y lista en principal. Ambos apuntan a la misma dirección de memoria, lista original.

Ejercicio 10 I

Construye una función en Python que encuentre todos los números reproductores de Fibonacci de dos y tres dígitos.

A saber: un número n se dice reproductor de Fibonacci si es capaz de reproducirse a sí mismo en una secuencia generada con los m dígitos del número en cuestión (sin alterar su orden) y continuando en la serie con un número que es la suma de los m términos precedentes.

Ejemplos:

- 47 es un número reproductor de Fibonacci pues la serie:
  4, 7, 11, 18, 29, 47, … contiene el 47.
- 13 no es un número reproductor de Fibonacci pues la serie:
  1, 3, 4, 7, 11, 18, … no contiene el 13.
- 197 es un número reproductor de Fibonacci pues la serie:
  1, 9, 7, 17, 33, 57, 107, 197, … contiene el 197.
- 985 no es un número reproductor de Fibonacci pues la serie:
  9, 8, 5, 22, 35, 62, 119, 216, 397, 732, 1345,… no contiene el 985.

Ejercicio 10 II

a. Construye una función es_repFib para comprobar si un número de cualquier longitud es reproductor de Fibonacci. La función recibe un número y devuelve un booleano.

b. Construye una función repFib2_3 para explorar todos los números reproductores de Fibonacci de dos y tres dígitos. La función no recibe nada y devuelve cuantos son y cuáles.

Ejercicio 10a

cap5_10a.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_10a.py (3.10.7)

File Edit Format Run Options Window Help

```
1  #PB Nov 2022
2  #Trabajar con listas.
3  def es_repFib(n):
4      ''' n es de cualquier longitud y se devuelve booleano'''
5      s=0
6      ld=[]
7      for d in str(n):
8          ld.append(d)
9          s+=int(d)
10 #          print('ld,s',ld,s)
11     while s<n:
12         ld.append(s)
13         ld=ld[1:]
14 #          print('ld,s',ld,s)
15         s=0
16         for d in ld:
17             s+=int(d)
18     if s==n:
19         return True
20     return False
```

>>>es_repFib(985)

True

Ejercicio 10b

cap5_10b.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_10b.py (3.10.7)

File Edit Format Run Options Window Help

```
#PB Nov 2022
#Trabajar con listas.
def repFib2_3():
    ''' se devuelve una tupla con contador y lista '''
    lista=[]
    c=0
    for n in range(10,1000):
        s=0
        ld=[]
        for d in str(n):
            ld.append(d)
            s+=int(d)
        while s<n:
            ld.append(s)
            ld=ld[1:]
            s=0
            for d in ld:
                s+=int(d)
        if s==n:
            lista.append(n)
            c+=1
    return c,lista
```

```
>>>repFib2_3()
(8, [14, 19, 28, 47, 61, 75, 197, 742])
```

Ejercicio 11

Crear dos funciones en Python para mezclar y ordenar dos listas ordenadas con dos métodos diferentes.

a. mezclando es una función que tiene como argumentos de entrada dos listas ordenadas con datos numéricos, y devuelve una única lista con todos sus elementos ordenados. Usar dos índices para recorrer las listas de entrada, no modificar las listas de entrada en el proceso.

b. ordena_mezclando es otra función que realiza la misma tarea que la anterior, pero comparando siempre los elementos de las posiciones 0 de ambas listas de entrada, es decir, eliminado los elementos de las listas a medida que se van volcando en la lista de salida.

Ejercicio 11a

cap5_11a.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_11a.py (3.10.7)

File Edit Format Run Options Window Help

```
1  #PB Nov 2022
2  def  mezclando(izquierda, derecha):
3    ordenada=[]
4    i, j = 0,0
5    while i < len(izquierda) and j < len(derecha):
6      if izquierda[i] <= derecha[j]:
7        ordenada.append(izquierda[i])
8        i=i+1
9      else:
10       ordenada.append(derecha[j])
11       j=j+1
12   ordenada +=izquierda[i:]
13   ordenada +=derecha[j:]
14   return ordenada
```

>>>mezclando([-1,9,9,21],[3.1,7,23,30,50])
[-1,3.1,7,9,9,21,23,30,50]

Ejercicio 11b

cap5_11b.py - C:\DATOS\Curso 2022-2023\GIQ\LAB_PYTHON_22_23\cap5_Funciones\resueltos\cap5_11b.py (3.10.7)

File Edit Format Run Options Window Help

```
1  #PB Nov 2022
2  def  ordena_mezclando(izda, dercha):
3    orden = []
4    while len(izda)!=0 and len(dercha)!=0:
5      if izda[0] <= dercha[0]:
6        orden.append(izda[0])
7        del(izda[0])
8      else:
9        orden.append(dercha[0])
10       del(dercha[0])
11   if len(izda)==0:
12     orden+=dercha[0:]
13   else:
14     orden+=izda[0:]
15   return orden
```

```
>>>ordena_mezclando([-1,9,9,21],[3.1,7,23,30,50])
[-1,3.1,7,9,9,21,23,30,50]
```

Módulos definidos por el programador

- Módulos ***predefinidos*** de Python, organizados en categorías: math, chempy, datetime, random, numpy, sqlite3, tkinter, html,...
- Módulos ***definidos por el programador***.
  - Reunir todas las funciones/procedimientos básicas para trabajar con un problema concreto en un módulo, archivo de Python.
  - Ventaja: evitar tener que añadir esas funciones/procedimientos en un programa cada vez que se necesiten.
  - Para usar esas funciones/procedimientos en un programa, basta importar el módulo en el mismo, con una sentencia del tipo:

    from *nombre_módulo* **import** *nombrefunción*

Ejemplo1. Crear un módulo con las funciones básicas sobre matrices: leer de teclado una matriz, escribirla en pantalla, inicializarla con ceros, construir la matriz identidad, calcular su traza, saber qué fila/columna está llena de ceros...

Ejemplo2. Crear un módulo con las funciones básicas para trabajar con listas: buscar, ordenar ascendente o descendentemente, calcular máximos y mínimos...

Ejemplo 1 I

mmatriz.py

File Edit For

```
1  def vermat(m):   #procedimiento
2      '''escribe matriz en pantalla, sólo elementos'''
3      for i in range(len(m)):
4          for j in range(len(m[0])):
5                          print(m[i][j],end=' ')
6          print()
7
8  def leermat(m):   #procedimiento
9      '''lee matriz de teclado'''
10     for i in range(len(m)):
11         for j in range(len(m[i])):
12             m[i][j]=float(input('Componente '+str(i+1)+','+str(j+1)+': '))
13
14 def matnula(fil,col): #funcion
15     '''crea matriz nula de filxcol'''
16     m=[]
17     for i in range(fil):
18        m.append([0]*col)
19     return m
20
21 def matidentidad(t): #función
22     '''construye matriz identidad de t x t'''
23     m=[[0 for j in range(t)] for i in range(t)]
24     for i in range(t):
25         m[i][i]=1
26     return m
```

Ejemplo 1 II

Para usar el módulo anterior mmatriz en un programa:

Escribir al principio del programa la sentencia:

from mmatriz **import** *

El módulo mmatriz.py debe estar almacenado en la misma carpeta que el programa que lo usa.