Programación
Licenciatura en Agroinformática
Licenciatura en Bioinformática

Tema 06: Funciones y Módulos

UNRaf UNIVERSIDAD NACIONAL DE RAFAELA

Divide y Venceras

- Un método para solucionar un problema complejo es dividirlo en subproblemas más sencillos.
- Normalmente, las partes en que se divide un programa deben poder desarrollarse independientemente entre sí.

El problema principal se soluciona resolviendo los subproblemas mediante subprogramas, conocidos como *funciones*.

Funciones en matemática

- En matemática, una función toma uno o más valores, llamados **argumentos**, y con ellos se calcula un **resultado**.
- Podemos definir una función matemática como:

$$f(x) = -4x^3 + 7x^2 + 13x - 42$$

- Luego de definida, podemos **evaluar** esa función, dándole un valor a la variable x:

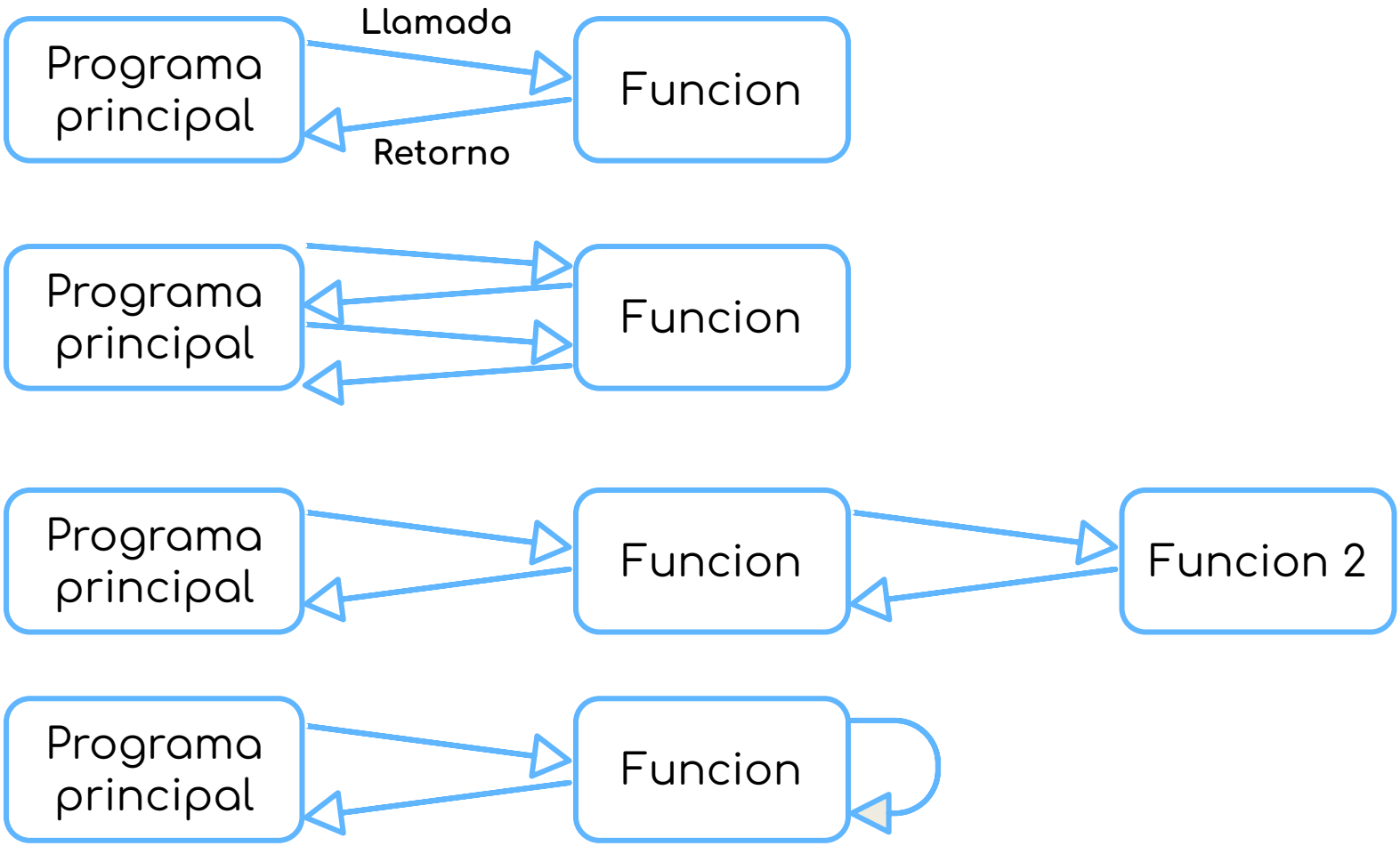
$$f(3) = -4 * 3^3 + 7 * 3^2 + 13 * 3 - 42$$

$$f(3) = -48$$

Funciones en programación

- Al igual que en matemática, en programación una **función** puede:
  - Recibir datos.
  - Realizar cálculos con esos datos.
  - Devolver resultados.
- Los datos para una función son proporcionados desde el programa principal.
- Una función puede llamar a otras funciones o a sí misma, para realizar cálculos necesarios para calcular sus propios resultados.

Esquema de llamadas



Tipos de funciones

- Existen dos tipos de funciones:
  - ***internas*** o ***built-in***
  - ***externas*** o definidas por el usuario.
- Las funciones internas son las que ya vienen incorporadas al lenguaje base (input, print, etc).
- Cuando las funciones internas no permiten realizar una operación deseada, se pueden construir funciones que la hagan.

Definiendo funciones

- Toda función cuenta con, al menos, 3 partes:
  - Un **nombre**, con el que se la llamará después.
  - **Parámetros** que son los datos que usará la función para realizar los cálculos.
  - Las **acciones** o **cálculos** que realiza la función con los datos.

Definiendo funciones

- También puede contar con otros dos elementos:
  - Un **resultado** que se devuelve al programa principal
  - **Variables propias** que puede necesitar para realizar los cálculos.
- Estos elementos no son obligatorios, pero sí es aconsejable usarlos.

Ejemplo de función

```
# Problema: calcular el perímetro de un círculo
# Definición de la función
def calculo_perimetro (r):
    pi = 3.1415
    per = 2 * pi * r
    return per

# Programa principal
radio = int(input("Ingrese un radio: "))
perimetro = calculo_perimetro(radio)
print (perimetro)
```

Ejemplo de función

```
# Problema: calcular el perímetro de un círculo
# Definición de la función
def calculo_perimetro (r):
    pi = 3.1415
    per = 2 * pi * r
    return per

# Programa principal
radio = int(input("Ingrese un radio: "))
perimetro = calculo_perimetro(radio)
print (perimetro)
```

La palabra clave ***def*** indica que lo que sigue es una función.

Ejemplo de función

```
# Problema: calcular el perímetro de un círculo
# Definición de la función
def calculo_perimetro (r):
    pi = 3.1415
    per = 2 * pi * r
    return per

# Programa principal
radio = int(input("Ingrese un radio: "))
perimetro = calculo_perimetro(radio)
print (perimetro)
```

Luego de **def** se indica el ***nombre*** de la función. Este nombre será el que se invoque en el programa principal

Ejemplo de función

```
# Problema: calcular el perímetro de un círculo
# Definición de la función
def calculo_perimetro (r):
    pi = 3.1415
    per = 2 * pi * r
    return per

# Programa principal
radio = int(input("Ingrese un radio: "))
perimetro = calculo_perimetro(radio)
print (perimetro)
```

Entre paréntesis se indican todos los *parámetros* de la función.

Ejemplo de función

```
# Problema: calcular el perímetro de un círculo
# Definición de la función
def calculo_perimetro (r):
    pi = 3.1415
    per = 2 * pi * r
    return per

# Programa principal
radio = int(input("Ingrese un radio: "))
perimetro = calculo_perimetro(radio)
print (perimetro)
```

Las *variables* que se definen dentro de una función se denominan "variables locales".

Ejemplo de función

```
# Problema: calcular el perímetro de un círculo
# Definición de la función
def calculo_perimetro (r):
    pi = 3.1415
    per = 2 * pi * r
    return per

# Programa principal
radio = int(input("Ingrese un radio: "))
perimetro = calculo_perimetro(radio)
print (perimetro)
```

La palabra clave *return* indica qué valor va a ser devuelto al programa principal y también marca el fin de la función.

Ejemplo de función

```
# Problema: calcular el perímetro de un círculo
# Definición de la función
def calculo_perimetro (r):
    pi = 3.1415
    per = 2 * pi * r
    return per

# Programa principal
radio = int(input("Ingrese un radio: "))
perimetro = calculo_perimetro(radio)
print (perimetro)
```

Al llamar una función en el código principal, hay que respetar el número y orden de parámetros con los que se la definió.

Ejemplo de función

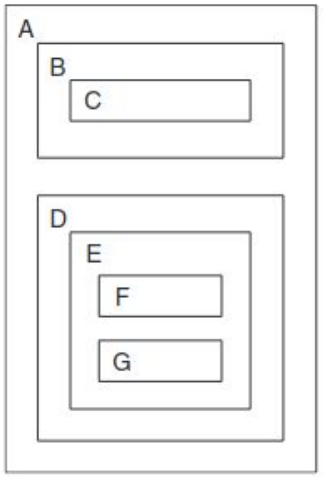
```
# Problema: calcular el perímetro de un círculo
# Definición de la función
def calculo_perimetro (r):
    pi = 3.1415
    per = 2 * pi * r
    return per

# Programa principal
radio = int(input("Ingrese un radio: "))
perimetro = calculo_perimetro(radio)
print (perimetro)
```

Variable donde se guardan los resultados de la función.

Variables locales vs globales

- Las **variables globales** son aquellas definidas en el ámbito del programa principal; las **variables locales** son aquellas definidas en el ámbito de una función.
- Una variable global puede ser accedida desde cualquier parte del programa, mientras que las variables locales **sólo** pueden accederse desde el ámbito donde fueron definidas



Variables definidas en	Accesibles desde
A	A, B, C, D, E, F, G
B	B, C
C	C
D	D, E, F, G
E	E, F, G
F	F
G	G

Ejemplo de ámbitos

- Construya 2 funciones que permitan calcular el tiempo que tarda un objeto en caer en la Tierra y en Marte, desde una altura indicada por el usuario. Considera la aceleración de la gravedad en la Tierra como 9.81 m/s^2 y en Marte como 3.72 m/s^2
- La fórmula de caída libre es:

$$y = H_0 - (g * t^2)/2$$

- Despejando el tiempo nos queda:

$$t = (2*H_0/g)^{½}$$

Ejemplo de ámbitos

```
def caida_tierra (altura):
    gt = 9.81
    t = (2 * altura / gt) ** 0.5
    return t

def caida_marte (altura):
    gm = 3.72
    t = (2 * altura / gm) ** 0.5
    return t
```

Ejemplo de ámbitos

```
def caida_tierra (altura):
    gt = 9.81
    t = (2 * altura / gt) ** 0.5
    return t

def caida_marte (altura):
    gm = 3.72
    t = (2 * altura / gm) ** 0.5
    return t
```

Variables locales

Ejemplo de ámbitos

```
def caida_tierra (altura):
    gt = 9.81
    t = (2 * altura / gt) ** 0.5
    return t

def caida_marte (altura):
    gm = 3.72
    t = (2 * altura / gm) ** 0.5
    return t

altura = 20
tiempo1 = caida_tierra (altura)
tiempo2 = caida_marte (altura)
```

Variables globales

Ejemplo de ámbitos

```
def caida_tierra (altura):
    gt = 9.81
    t = (2 * altura / gt) ** 0.5
    return t

def caida_marte (altura):
    gm = 3.72
    t = (2 * altura / gm) ** 0.5
    return t

altura = 20
tiempo1 = caida_tierra (altura)
tiempo2 = caida_marte (altura)

print (gt, gm)
```

Intentar acceder a variables locales fuera de su ámbito da error!!!

Return

- En Python, una función va a modificar todas las variables sin salir de su ámbito.
- Para poder utilizar el resultado calculado por una función en el programa principal se debe incluir la sentencia **return**.
- Si se indica un **return** en la función, es necesario asignar el resultado

Return

```
def mitad (numero):
    numero /= 2


def mitad2 (numero):
    numero /= 2
    return numero


num = 10
print("Al comenzar:",num)
mitad(num)
print("Primer funcion:",num)
num = mitad2(num)
print("Segunda funcion:",num)
```

```
(base) matias@rfgenom002:~$ python test.py
Al comenzar:  10
Primer funcion:  10
Segunda funcion:  5.0
```

Parámetros por defecto

- Es posible asignar valores por defecto al definir una función.
- Si al llamar la función, no se especificó ningún valor, se usará el que se estableció como por defecto.

```
def caida (altura, g=9.81):
    t = (2 * altura / g) ** 0.5
    return t


caida (20)        # Se usará 9.81 como valor de g
caida (20, 3.72)# En este caso, g va a valer 3.72
```

Funciones y listas

- El comportamiento de las listas es distinto al de variables primitivas.
- Si una lista es modificada en una función, ese cambio se refleja en la variable general, por más que no se indique un return

```
def agrega (lista, valor):
    lista.append(valor)

lista = ["a", "b", "c"]
print (lista) # Se obtiene "a", "b", "c"
valor = "d"
agregar(lista, valor)
print (lista) # "a", "b", "c", "d"
```

¿Dónde defino cada cosa?

- En Python, para utilizar una función, debe ser definida antes de ser llamada.
- Una función puede estar definida en el medio del programa principal, pero esto **no** es aconsejable.
- En general, se recomienda definir todas las funciones al comienzo del código; luego las variables globales y constantes que sean necesarias y por último el código del programa principal.

Módulos

- Muchas funciones comúnmente usadas (por ej: funciones trigonométricas) no están incorporadas por defecto en Python.
- Si uno quisiera usar esas funciones, debería implementarlas manualmente.
- También es posible utilizar las funciones ya implementadas por otras personas.
- Para ello, es necesario descargar e instalar paquetes de funciones llamados **módulos**.

Módulos II

- Un módulo es un conjunto de funciones y variables, definidas por el usuario, que se complementan a las funciones internas.
- Para poder cargar un módulo se utilizan las palabras claves ***import*** y ***from***.
- Para usar una función dentro del módulo, se la invoca usando el nombre del módulo, un punto y el nombre de la función.

Ejemplo de uso de módulos

```
import numpy
angulo = 30
angulo_radianes = numpy.radians(angulo)
coseno = numpy.cos(angulo_radianes)
print (coseno)
seno = numpy.sin(angulo_radianes)
print (seno)

import random
print(random.randrange(0,100))

import sys
print(sys.path)
print(sys.path[0])
print(sys.argv)
```

Ejemplo 1

- Implemente una función que permita calcular el promedio de una lista de valores.

Entrada y salidas
i: Lista de valores
o: promedio

Pseudocódigo:

```
recibir lista
suma = 0
for i in range(0:len(lista)):
    suma += lista[i]
devolver (suma/len(lista))
```

Ejemplo 1

```
def promedio (lista):
    suma = 0
    for i in range (0, len(lista)):
        suma += lista[i]
    return suma/len(lista)

lista = [2, 4, 6, 8, 10, 13]
prom = promedio(lista)
print (prom)
```

Ejemplo 2

- Construya una función que permita calcular el valor de un producto sumándole el IVA. A menos que se explicite, el IVA se considera el 21%.

Entrada y salidas
i: precio
o: precio + IVA

Pseudocódigo:
recibir precio e IVA (por defecto, 21)
devolver precio + precio * (IVA/100)

Ejemplo 2

```
def precio_nuevo (precio, iva=21):
    precio = precio + precio * (iva/100)
    return precio

precio = 1000
IVA = 10.5
iva_105 = precio_nuevo(precio, IVA)
print (iva_105)
iva_21 = precio_nuevo(precio)
print (iva_21)
```

Ejemplo 3

- Diseñe una función que simule el lanzamiento de una moneda

Ejemplo 3

- Diseñe una función que simule el lanzamiento de una moneda

Entrada y salidas
i: ninguna
o: cara o ceca

Pseudocódigo:
calcular un número entero al azar entre 0 y 1.
si numero == 0:
    devolver “cara”
sino:
    devolver “ceca”

Ejemplo 3

```
import random

def cara_ceca ():
    valor = random.randrange(0,2)
    if valor == 0:
        return "cara"
    else:
        return "ceca"

print ("Intento 1: ", cara_ceca())
print ("Intento 2: ", cara_ceca())
print ("Intento 3: ", cara_ceca())
```

Ambitos de datos primitivos

```
def modificar_elemento(elemento1, elemento2):
    elemento1 = elemento2

def modificar_elemento_2(elemento1, elemento2):
    elemento1 = elemento2
    return elemento1

lista = ["John", "Paul", "George", "Ringo"]
quinto = "Yoko"

print("Modificar elemento 1")
print("Antes de ejecutar la funcion: ",quinto)
modificar_elemento(quinto, "Apu")
print("Despues de ejecutar la funcion: ",quinto)

print("Modificar elemento 2")
print("Antes de ejecutar la funcion: ",quinto)
quinto = modificar_elemento_2(quinto, "Apu")
print("Despues de ejecutar la funcion: ",quinto)
```

Ambitos de datos primitivos

```
def modificar_elemento(elemento1, elemento2):
    elemento1 = elemento2

def modificar_elemento_2(elemento1, elemento2):
    elemento1 = elemento2
    return elemento1

lista = ["John", "Paul", "George", "Ringo"]
quinto = "Yoko"

print("Modificar elemento 1")
print("Antes de ejecutar la funcion: ",quinto)
modificar_elemento(quinto, "Apu")
print("Despues de ejecutar la funcion: ",quinto)

print("Modificar elemento 2")
print("Antes de ejecutar la funcion: ",quinto)
quinto = modificar_elemento_2(quinto, "Apu")
print("Despues de ejecutar la funcion: ",quinto)
```

Ambitos de datos primitivos

```
(base) matias@rfgenom002:~$ python test.py
Modificar elemento 1
Antes de ejecutar la funcion:  Yoko
Despues de ejecutar la funcion:  Yoko
Modificar elemento 2
Antes de ejecutar la funcion:  Yoko
Despues de ejecutar la funcion:  Apu
```

Ambitos de datos abstractos

```
def agregar_elemento(lista, elemento):
    lista.append(elemento)

def agregar_elemento_2(lista, elemento):
    lista.append(elemento)
    return lista

lista = ["John", "Paul", "George", "Ringo"]
quinto = "Apu"

print("Agregar elemento")
print("Antes de ejecutar la funcion: ",lista)
agregar_elemento(lista, "Apu")
print("Despues de ejecutar la funcion: ",lista)

lista2 = []
print("Agregar elemento 2")
print("lista antes de ejecutar la funcion: ",lista)
print("lista2 antes de ejecutar la funcion: ",lista2)
lista2 = agregar_elemento_2(lista, "Apu")
print("lista despues de ejecutar la funcion: ",lista)
print("lista2 despues de ejecutar la funcion: ",lista2)
```

Ambitos de datos abstractos

```
def agregar_elemento(lista, elemento):
    lista.append(elemento)

def agregar_elemento_2(lista, elemento):
    lista.append(elemento)
    return lista

lista = ["John", "Paul", "George", "Ringo"]
quinto = "Apu"

print("Agregar elemento")
print("Antes de ejecutar la funcion: ",lista)
agregar_elemento(lista, "Apu")
print("Despues de ejecutar la funcion: ",lista)

lista2 = []
print("Agregar elemento 2")
print("lista antes de ejecutar la funcion: ",lista)
print("lista2 antes de ejecutar la funcion: ",lista2)
lista2 = agregar_elemento_2(lista, "Apu")
print("lista despues de ejecutar la funcion: ",lista)
print("lista2 despues de ejecutar la funcion: ",lista2)
```

Ambitos de datos abstractos

```
(base) matias@rfgenom002:~$ python test.py
Agregar elemento
Antes de ejecutar la funcion:  ['John', 'Paul', 'George', 'Ringo']
Despues de ejecutar la funcion:  ['John', 'Paul', 'George', 'Ringo', 'Apu']
Agregar elemento 2
lista antes de ejecutar la funcion:  ['John', 'Paul', 'George', 'Ringo', 'Apu']
lista2 antes de ejecutar la funcion:  []
lista despues de ejecutar la funcion:  ['John', 'Paul', 'George', 'Ringo', 'Apu', 'Apu']
lista2 despues de ejecutar la funcion:  ['John', 'Paul', 'George', 'Ringo', 'Apu', 'Apu']
```

Modulo *sys*

- Permite acceder a parámetros y funciones directamente relacionadas con el sistema.
- Son funciones muy útiles para modificar el comportamiento de un script al momento de ejecutarlo o encontrar información del sistema.

sys.argv

- Devuelve la lista de argumentos pasados al *script* al momento de ejecutarlo.
- sys.argv[0] contiene el nombre del archivo ejecutándose.

```
import sys

argumentos = sys.argv
print(argumentos[0])
print(argumentos[1])
print(argumentos[2])
suma =
float(argumentos[1])+float(argumentos[2])
print(suma)
```

```
(base) matias@rfgenom002:~$ python test.py 10 20
test.py
10
20
30.0
```

Analizando códigos I (1)

```
import random
import sys

def funcion_sin_nombre():
    numero = random.randrange(0,3)
    if numero == 0:
        return "piedra"
    elif numero == 1:
        return "papel"
    else:
        return "tijera"
```

Analizando códigos I (2)

```
def comparar(elemento1, elemento2):
    if elemento1 == "piedra" and elemento2 == "piedra":
        return 0
    elif elemento1 == "piedra" and elemento2 == "papel":
        return 2
    elif elemento1 == "piedra" and elemento2 == "tijera":
        return 1
    elif elemento1 == "papel" and elemento2 == "piedra":
        return 1
    elif elemento1 == "papel" and elemento2 == "papel":
        return 0
    elif elemento1 == "papel" and elemento2 == "tijera":
        return 2
    elif elemento1 == "tijera" and elemento2 == "piedra":
        return 2
    elif elemento1 == "tijera" and elemento2 == "papel":
        return 1
    else:
        return 0
```

Analizando códigos I (3)

```
apuesta_usuario = sys.argv[1]
apuesta_compu = funcion_sin_nombre()
resultado = comparar(apuesta_usuario, apuesta_compu)
print("Apuesta usuario: ", apuesta_usuario)
print("Apuesta computadora: ", apuesta_compu)
if resultado == 1:
    print("Ganador: usuario")
elif resultado == 2:
    print("Ganador: computadora")
else:
    print("Empate")
```

Analizando códigos II

```
import numpy
import sys

def otra_funcion_sin_nombre(v0, alpha):
    t = (2 * v0 * numpy.sin(alpha)) / 9.81
    xmax = v0 * numpy.cos(alpha) * t
    ymax = (v0**2 * numpy.sin(alpha)**2) / (2 * 9.81)
    return xmax, ymax

v = float(sys.argv[1])
a = float(sys.argv[2])
a = numpy.radians(a)
x,y = otra_funcion_sin_nombre(v,a)
print(x)
print(y)
```