

Funciones y Módulos

...

Programación
2024

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:

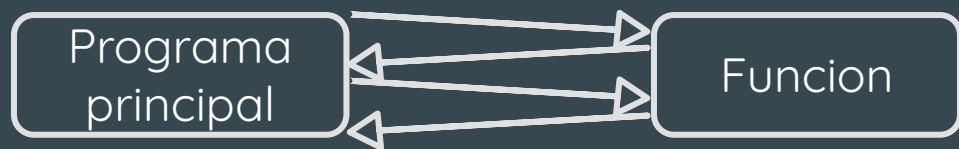
$$f(3) = -4 * 3^3 + 7 * 3^2 + 13 * 3 - 42$$

$$f(3) = -48$$

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
```

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

```
# 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
```

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

```
# 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
```

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

```
# 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
```

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

```
# 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 reservada **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

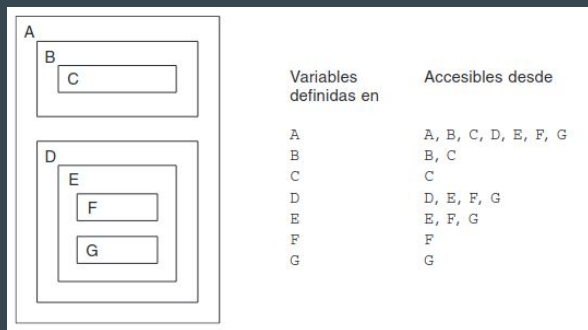
```
# Problema: calcular el perímetro de un círculo
# Definición de la función
def calculo_perimetro (r):
    pi = 3.1415
    per = 2 * pi * r
    return per
```

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

Los resultados de la función se pueden guardar en una variable.

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.
- Las variables pueden ser accedidas desde el ámbito que se las creó y en todos los ámbitos interiores.



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/s² y en Marte como 3.72 m/s²
- La fórmula de caída libre es:

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

- Despejando el tiempo nos queda:

$$t = (2 * H_0 / g)^{1/2}$$

Ejemplo de función

```
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)
```

Ejemplo de función

```
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
locales

Ejemplo de función

```
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 función

```
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**.

Ejemplo de función

```
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 variables mutables 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) # No hay return!!
```

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

¿Dónde definir 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** y luego el código del programa principal.

Módulos

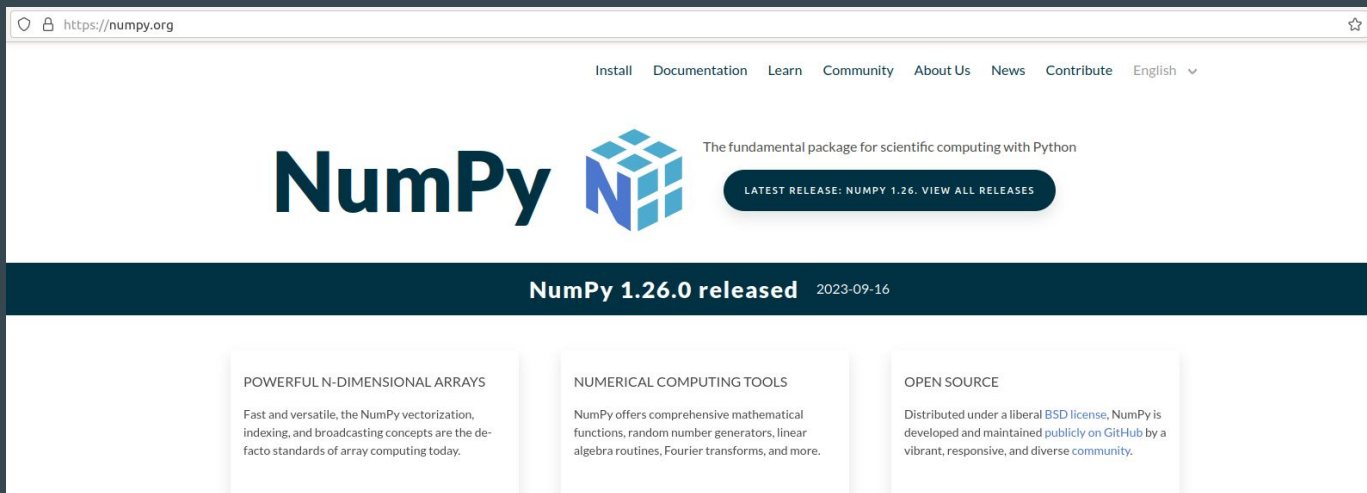
- Muchas funciones comúnmente usadas (por ej: funciones trigonométricas) no están incorporadas por defecto en Python.
- Si uno quisiera usar esas funciones, debería implementarlas manualmente o utilizar las funciones ya implementadas por otras personas.
- Para ello, es necesario descargar e instalar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

Módulos

- 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 la palabra clave **import**.
- Para usar una función dentro del módulo, se la invoca usando el **nombre del módulo, un punto y el nombre de la función**.

Algunos módulos: *numpy*

- Es uno de los módulos más utilizados de Python.
- Tiene numerosas funciones matemáticas y tipos de datos específicos implementados.




Algunos módulos: *numpy*






```
import numpy
```

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

Algunos módulos: *matplotlib*

- Es la principal herramienta de graficación de Python.
- Existen muchos tipos de gráficos.
- Cada elemento dentro del gráfico puede modificarse a gusto.

 [Plot types](#) [User guide](#) [Tutorials](#) [Examples](#) [Reference](#) [Contribute](#) [Releases](#)

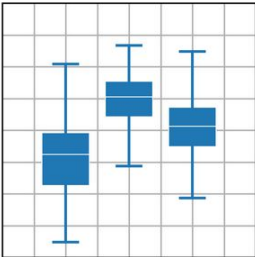
    

Matplotlib: Visualization with Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats.
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.

Try Matplotlib (on Binder) →



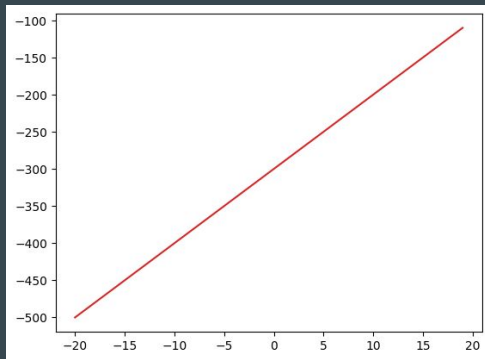
boxplot(X)

Algunos módulos: *matplotlib*

```
import matplotlib.pyplot as plt
# Como "matplotlib.pyplot" es muy largo, se puede
# definir un alias con "as" para evitar escribir
# todo el nombre

x = range(-20, 20)
a = 10
b = -300
y = []
for i in x:
    resultado = a * i + b
    y.append(resultado)

plt.plot(x, y, color = 'tab:red')
plt.show()
```



Algunos módulos: *random*

- Permite generar numeros (pseudo)aleatorios.
- Se puede utilizar para:
 - Generar numeros en rangos de valores enteros.
 - Generar numeros en rangos de valores reales.
 - Elegir valores en una secuencia.
- Los valores son calculados con una fórmula matemática que parte de un valor “semilla”.
- Si la semilla es la misma, la fórmula va a dar los mismos resultados (reproducibilidad!)

Algunos módulos: *random*

```
import random
print("Enteros")
print(random.randrange(-1000, 1000))
print(random.randrange(-1000, 1000))
print(random.randrange(-1000, 1000))
```

Algunos módulos: *random*

```
random.seed()
```

```
print("Flotantes 0-1")  
print(random.random())  
print(random.random())  
print(random.random())
```

```
print("Flotantes")  
print(random.uniform(-1000, 1000))  
print(random.uniform(-1000, 1000))  
print(random.uniform(-1000, 1000))
```

```
print("Vocales")  
seq = ["a", "e", "i", "o", "u"]  
print(random.choice(seq))  
print(random.choice(seq))  
print(random.choice(seq))
```

Algunos módulos: *sys* y *os*

- Permiten acceder a variables y realizar acciones sobre el sistema operativo.
- Un uso muy común del módulo *sys* es el paso de argumentos a la hora de ejecutar un *script*.
- Esto permite escribir un solo programa y utilizarlo con distintos datos sin necesidad de alterar el código.
- El módulo *os* puede utilizarse para listar los archivos en una ruta, crear carpetas y eliminar archivos (usar con precaución!!!)

Algunos módulos: *sys*

```
import sys
```

```
argumentos = sys.argv # Forma una lista con todos los datos de ejecución
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
```

Algunos módulos: os

```
import os
```

```
ruta_actual = os.getcwd()  
print(ruta_actual)
```

```
archivos = os.listdir()  
print(archivos)
```

```
if os.path.exists("archivo_para_borrar.txt"):  
    os.remove("archivo_para_borrar.txt")
```

```
archivos = os.listdir()  
print(archivos)
```

```
dummy@rfgenom001:~/Documentos$ python3 script.py  
/home/dummy/Documentos  
['Programas', 'archivo_para_borrar.txt', 'ejercicios.py', 'script.py']  
['Programas', 'ejercicios.py', 'script.py']
```

Analizando códigos I (1)

```
import random
import sys

def funcion_sin_nombre():
    numero = random.randrange(0,3)
    if numero == 0:
        return "pi"
    elif numero == 1:
        return "pa"
    else:
        return "ti"
```


Analizando códigos I (2)

```
def comparar(elemento1, elemento2):  
    if elemento1 == "pi" and elemento2 == "pi":  
        return 0  
    elif elemento1 == "pi" and elemento2 == "pa":  
        return 2  
    elif elemento1 == "pi" and elemento2 == "ti":  
        return 1  
    elif elemento1 == "pa" and elemento2 == "pi":  
        return 1  
    elif elemento1 == "pa" and elemento2 == "pa":  
        return 0  
    elif elemento1 == "pa" and elemento2 == "ti":  
        return 2  
    elif elemento1 == "ti" and elemento2 == "pi":  
        return 2  
    elif elemento1 == "ti" and elemento2 == "pa":  
        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)
```