

# **FUNCIONES**

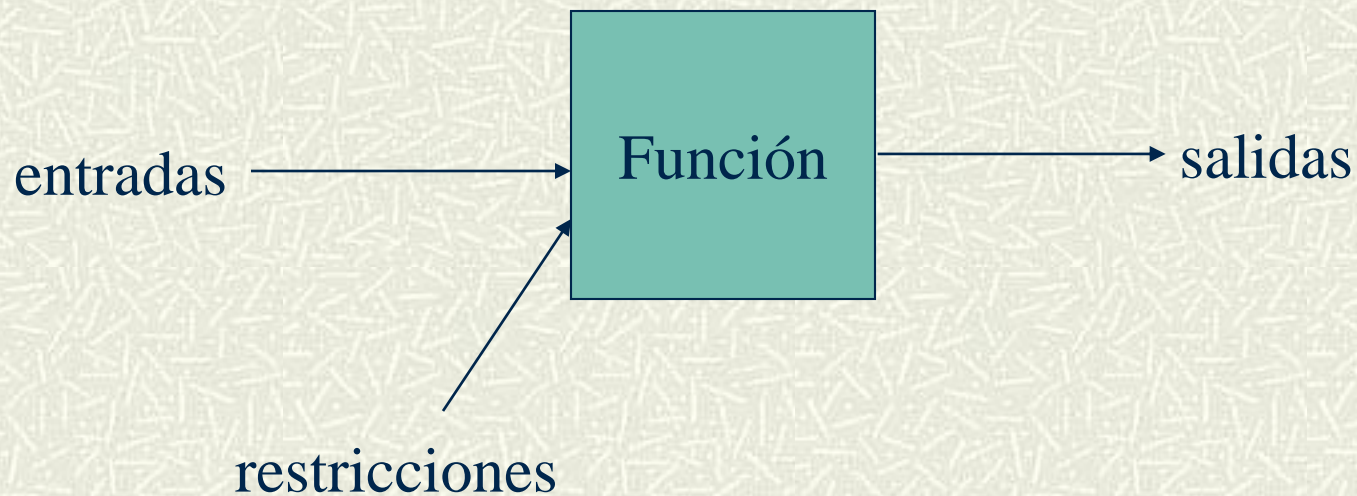


# **FUNCIONES**

---

- # Generalmente un programa está construido por un grupo de funciones (subprogramas, procedimientos)
- # Función: conjunto de instrucciones que realizan una tarea específica
- # Cada función realiza una parte del programa, el resultado de cada función contribuye a obtener los resultados completos del programa. 2 / 45

## # Modelo general de una función





## # Tipos


- Preconstruídas (Built-in functions)
- Desarrolladas por el programador

# Funciones preconstruídas (Built-in functions)

---

- # Funciones que tiene el lenguaje
- # Vienen incorporadas con el lenguaje
- # Están disponibles para uso del programador



- 
- # Para usar una función la debemos llamar: ponemos su nombre y entre paréntesis le enviamos los argumentos, es decir, los valores que ocupa para realizar la tarea

`nombre_función(valor1, valor2, ...)`

- # La función se ejecuta y nos da (retorna) los resultados

# Ejemplos

---

# abs(expresión)

- Valor absoluto

```
>>> abs(-3)
```

```
3
```

# pow(expresión1, expresión2)

- Elevar la base expresión1 a la potencia expresión2

```
>>> pow(2, 4)
```

```
16
```


- # Se puede usar cualquier expresión como argumento
- # Recuerde: cuando hay una expresión, primero se calcula el valor de la misma y luego se usa el valor calculado

```
>>> x = 2
```

```
>>> pow(5, x+1)
```

```
125
```





# En caso de ocupar el resultado de una función para procesos posteriores, este resultado se asigna a una variable

```
>>> y = pow(5, 3)
```

```
>>> x1 = y+2
```

```
>>> x2 = y-2
```

# Funciones para conversión de tipos de datos

---

- # Python provee un conjunto de funciones que convierten datos de un tipo de datos a otro tipo de datos

## # Función *int*

- Convierte valores numéricos o strings con solo números enteros a un valor numérico entero. Si tiene decimales los elimina (trunca)

```
>>> int(3.9)
```

```
3
```

```
>>> int(-5.3)
```

```
-5
```

```
>>> int("32")
```

```
32
```

```
>>> int("val")
```

# también da error si el string no es un número entero

**Traceback (most recent call last):**

**File "<pyshell#20>", line 1, in <module>**

**int("val")**

**ValueError: invalid literal for int() with base 10: 'val'**

## # Función *float*

- Convierte valores numéricos o strings con solo números a un valor numérico flotante

```
>>> float(3.1421)
```

```
3.1421
```

```
>>> float(2)
```

```
2.0
```

```
>>> float("89.23")
```

```
89.23
```



## # Función *str*

- Convierte cualquier tipo de datos al tipo string

```
>>> str(123)
```

```
'123'
```

```
>>> str(3.14159)
```

```
'3.14159'
```

```
>>> str("programación")
```

```
'programación'
```

```
>>> str(True)
```

```
'True'
```

# Función para leer datos

## # Función **input([prompt])**


- Cuando el programa encuentra esta instrucción detiene la ejecución para esperar que le den un valor por medio del teclado
- La función es para darle datos al programa
- Si se especifica el prompt este es desplegado en el dispositivo de salida (normalmente el monitor)
- El prompt guía al usuario sobre el dato que va a dar
- El valor es leído como un dato de tipo string

```
>>> materia = input("Materia que estamos estudiando: ")
```

```
Materia que estamos estudiando: programación
```

```
>>> materia
```

```
'programación'
```



# Si ocupamos que el dato sea numérico entero entonces convertimos el string leído a ese tipo de datos

```
>>> valor = int(input("Número: "))
```

```
Número: -5
```

```
>>> valor
```

```
-5
```

# Función para imprimir datos

---

# Función **print([object, ...])** ← forma básica

■ Imprimir o desplegar resultados

```
>>> valor = int(input("Número: "))  
Número: -5
```

```
>>> print("El valor absoluto de", valor, " es", abs(valor))  
El valor absoluto de -5 es 5
```



# Funciones dentro de funciones

---

# Se refiere a que una función puede tener como argumentos el resultado de otras funciones

# Ejemplo

```
>>> y = pow(4, abs(a+2))
```

# FUNCIONES CONSTRUIDAS POR EL PROGRAMADOR

# Definición de funciones del programador

```
def nombre_función (parámetros):  
    cuerpo de la función
```

Ejemplo de una definición de función construida por el programador: recibe un valor, calcula una fórmula y da como resultado ese cálculo

```
def convertir_c_f(grados_celsius):  
    grados_fahrenheit = grados_celsius * 9/5 + 32  
    return grados_fahrenheit
```

- # A cada función le asignamos un nombre
- # Los nombres de las funciones siguen las mismas reglas de los nombres de variables
- # Por lo general la definición de las funciones tienen parámetros, es decir, nombres de variables que van a contener los valores necesarios para realizar el proceso y así obtener resultados
- # El cuerpo de la función son las instrucciones que pertenecen a la función y van indentadas (sangría) respecto a la palabra reservada **def**

## # IMPORTANCIA DE LA INDENTACIÓN EN PYTHON (sangría, tabulación):

- Delimita los bloques de instrucciones que pertenecen a determinadas estructuras, como por ejemplo en la definición de funciones



# EJECUCIÓN DE FUNCIONES

- Después de definir una función la podemos usar, para ello hay que llamarla: ponemos su nombre y entre paréntesis los valores (argumentos) que ocupamos enviarle para realizar la tarea

**nombre\_función(valor1, valor2, ...)**

- Sino tiene argumentos se pone el nombre y los paréntesis
- Los argumentos se asignan a los parámetros en la definición de la función en una relación de 1 a 1 de izquierda a derecha
- Los parámetros son las variables en las cuales la función recibe los argumentos

**def nombre\_función(parámetro1, parámetro2, ...):**

# definición de la función

```
def convertir_c_f(grados_celsius):
```

```
    grados_fahrenheit = grados_celsius * 9/5 + 32
```

```
    return grados_fahrenheit
```

# programa principal: aquí empieza la ejecución del programa

# usar la función: llamar a la función por su nombre

```
gc = 25
```

```
gf = convertir_c_f(gc) # asignar el valor calculado a una variable
```

```
print(gf)
```

# otra forma de usar la función

```
print(convertir_c_f(gc) ) # imprimir directamente el valor retornado
```

- # La **definición** de una función no se ejecuta, es decir, NO altera el flujo de ejecución del programa
- # La **ejecución** de funciones SI altera el flujo de ejecución del programa:
  - Cuando una función es llamada, el flujo de ejecución se traslada a la función
  - Cuando la función termina, el flujo de ejecución continúa a la instrucción desde donde fue llamada



# return

---

- # Cada vez que se ejecuta una función se crean en memoria sus parámetros
- # Cuando la función termina de ejecutarse todos los parámetros y variables creadas en dicha función son borradas de la memoria
- # Por eso, si posteriormente a la finalización de la función ocupamos valores calculados en dicha función estos deben ser retornados a la instrucción que hizo la llamada

***return*** [ expresión1, ... , expresiónN]



# EJEMPLO: FUNCIÓN PARA CALCULAR ÁREA DE UN RECTÁNGULO

---

# definición de la función

```
def area_rectangulo(base, altura):
```

```
    area = base * altura
```

```
    return área
```


# programa principal

# usar la función

```
a = area_rectangulo(5, 12)
```

```
print("Área del rectángulo:", a)
```

25/45



# La ejecución de una función puede terminar bajo dos condiciones:

- Cuando encuentre la instrucción **return**
- Cuando no tenga instrucciones por ejecutar

# Si el return no indica alguna expresión o la función termina sin el uso de esta instrucción, automáticamente se retorna el valor especial llamado **None** (Nada)

# ALCANCE DE LAS VARIABLES

---

# El alcance de las variables indica en qué partes del programa se pueden usar las variables creadas

# Tipos de alcance

- Variables locales
- Variables globales

27/45

## # Variables locales

- Los parámetros de las funciones
- Las variables que se crean dentro de una función: en asignaciones
- Se pueden usar solo dentro de la función: cuando la función termina de ejecutarse todas las variables locales se borran. Por eso los valores que necesitan ser preservados después de la ejecución de la función deben retornarse.
- No pueden ser usadas fuera de ella



```
def suma(n1,n2):  
    resul=n1+n2  
    print(resul)  
    return
```

```
# programa principal  
num1=4  
num2=5  
suma(num1,num2)  
print("Fin")  
>>>  
9  
Fin
```

**Cuando termina la ejecución de la función suma, las variables n1, n2 y resul desaparecen de la memoria, sus valores se pierden.**

**Por ello, en caso de ocupar posteriormente en el programa los valores calculados en una función debemos retornarlos.**



---

Las variables definidas dentro de una función no se pueden usar fuera de la función

```
>>> print (resul)
```

Traceback (most recent call last):

File "<pyshell#40>", line 1, in <module>

print resul

NameError: name 'resul' is not defined

## # Variables globales

- Variables creadas fuera de cualquier función
- Las variables globales se pueden usar en las funciones para efectos de referencia, no se pueden modificar
- Ejemplo:

```
def multiplica():
```

```
    m = n1 + n2
```

```
    print(n1, "x", n2, "=", m)
```

```
n1=10
```

```
n2= 9
```


```
multiplica()
```

# DOCUMENTACIÓN DEL PROGRAMA FUENTE

---

- # Son notas en lenguaje natural que ponemos en el programa para explicar lo que hace
- # Buena práctica en la construcción de programas: colocar en el código suficientes comentarios que permitan mejorar el entendimiento del programa
- # Los comentarios no son analizados por el lenguaje de programación, no tienen efecto sobre la ejecución del programa





# En Python podemos documentar el código de dos formas:

- Comentarios a nivel de línea
- Comentarios multilínea

33/45

## # Comentarios a nivel de línea

- Se establecen con el símbolo “#”

# Función que calcula el área de un rectángulo

# Entradas: base, altura

# Salidas: área del rectángulo

# Restricciones: base y altura son números mayores que cero

# Autor:

# Fecha de creación:

```
def area_rectangulo(base, altura):
```

```
    area = base * altura # cálculo del área
```

```
    return area
```

## # Documentación sugerida

- Al inicio de cada programa
  - Función del programa
  - Entradas
  - Salidas
  - Restricciones o limitaciones
  - Autor
  - Fecha de creación
  - Historial de modificaciones: Modificación/Autor/Fecha

## # Comentarios multilínea

- Se establecen con tres comillas (dobles o simples) al inicio y al final del comentario

```
def area_rectangulo(base, altura):
```

```
    """Función para calcular área de un rectángulo.
```

```
    Entradas: base y altura.
```

```
    Salidas: área del rectángulo.
```

```
    Restricciones: la base y la altura deben ser números  
    mayores que cero."""
```

```
    area = base * altura
```

```
    return area
```



## # Otra documentación sugerida

- En cada componente del programa, por ejemplo una función, se sugiere documentar lo que hace
- Cualquier otra parte del programa que tenga un aspecto importante o especial también debe ser documentada

# MÓDULOS DE PYTHON

---

- # Conjunto de funciones y otros elementos relacionados
- # Facilitan el desarrollo de programas ya que permite reutilizar partes que previamente se han construido
- # Python provee un conjunto de módulos preconstruidos
- # Cada módulo tiene un nombre

# *Ejemplo con módulo math:* módulo preconstruído que contiene funciones matemáticas

```
>>> import math  
>>> math.sqrt(25)  
5.0
```

Con esta forma de importar módulos hay que anteponerle a la función el nombre del módulo

## # **math** contiene

- Potencias y logaritmos
- Conversión de ángulos
- Factorial
- Constantes de manejo común: pi, e
- Otros elementos



## # Otros módulos

- `os`: funciones para interactuar con el sistema operativo, por ejemplo crear una carpeta o directorio: `mkdir`
- `time`: funciones para interactuar con la fecha y hora del sistema, por ejemplo obtener esos datos: `ctime`

## # Lista de elementos de un módulo

```
>>> import módulo
```

```
>>> dir(módulo)
```

```
>>> import math
```

```
>>> dir(math)
```

## # Documentación de los elementos de un módulo

```
>>> import módulo
```

```
>>> help(módulo)
```

```
>>> import math
```

```
>>> help(math)
```

## # Documentación de un elemento específico

```
>>> help(math.sqrt)
```

Help on built-in function sqrt in module math:

```
sqrt(...)
```

```
sqrt(x)
```

Return the square root of x.

## # Programa principal (main program, main function)

- La ejecución de cualquier programa empieza en el programa principal
- Es el punto de entrada para que inicie la ejecución de un programa
- El programa principal esta compuesto por todas las instrucciones que se pueden ejecutar de forma inmediata cuando el programa fuente es ejecutado. No necesitan ser activadas por ningún otro componente del programa para que puedan ser ejecutadas



“El diálogo, basado en sólidas leyes morales, facilita la solución de los conflictos y favorece el respeto de la vida, de toda vida humana.

Por ello, el recurso a las armas para dirimir las controversias representa siempre una derrota de la razón y de la humanidad.”

Juan Pablo II

