

Programación para Física y Astronomía

Departamento de Física.

Corodinadora: C Loyola

Profesores C Femenías / F Bugini / D Basantes

Primer Semestre 2025

Universidad Andrés Bello

Departamento de Física y Astronomía



Repaso y Contexto

Funciones en Python

Parámetros y Alcance

Módulos y Paquetes

Ejemplos y Actividad

Conclusiones

Repaso y Contexto

Repaso y Contexto ∈ Recapitulación de la Sesión Anterior (Sesión 6)

- **Semana 3, Sesión 2 (Sesión 6)** se centró en:
 - Laboratorio práctico con estructuras de control (`if`, `while`).
 - Problemas tipo: “Adivina el número”, “Calculadora de Calificaciones”, “Movimiento Discreto”.
 - Manejo de validaciones, contadores y salidas controladas de bucles.

Ejemplo: `def f(): x=10` — la variable `x` vive solo en `f()`.

- **Comprender** la sintaxis y concepto de funciones en Python (**def**, parámetros, **return**).
- **Explorar** cómo organizar código en módulos y paquetes simples.
- **Aplicar** estos conceptos a pequeños proyectos para mejorar reutilización y claridad.
- **Conectar** funciones con los temas previos (estructuras de control, manejo de datos, etc.).

Funciones en Python

- **Reutilización de Código:** evitar escribir la misma lógica en múltiples lugares.
- **Organización:** encapsular tareas específicas en “cajas negras”.
- **Legibilidad:** el nombre de la función describe la operación que realiza.
- **Mantenibilidad:** cambiar la lógica en un solo lugar (la definición de la función).

```
1 def nombre_de_funcion(param1, param2, ...):  
2     """  
3     Documentación opcional (docstring).  
4     Explica qué hace la función, los parámetros y el retorno.  
5     """  
6     # bloque de código  
7     # opcionalmente retornar un valor  
8     return resultado
```

- **Parámetros:** valores de entrada que la función utiliza.
- **return:** finaliza la función y opcionalmente devuelve un valor.

Funciones en Python ∈ Ejemplo: Función para Calcular el Área de un Círculo

```
1 import math
2
3 def area_circulo(radio):
4     """
5     Retorna el área de un círculo de radio 'radio'.
6     Formula:  $\pi * r^2$ 
7     """
8     area = math.pi * (radio**2)
9     return area
10
11 # Uso de la función
12 r = 5
13 a = area_circulo(r)
14 print("El área del círculo es:", a)
```

- Encapsulamos la operación en `area_circulo`.
- Parámetro `radio`.
- Retorno con `return area`.

Parámetros y Alcance

Parámetros y Alcance ∈ Parámetros Posicionales y Opcionales

```
1 def calcular_salario(base, horas_extra=0):  
2     """Calcula salario sumando base + 2000 por hora extra"""  
3     return base + 2000 * horas_extra  
4  
5 s1 = calcular_salario(50000)           # horas_extra por defecto:  
    ↪ 0  
6 s2 = calcular_salario(50000, 3)       # 3 horas extra  
7 print(s1, s2)
```

- Parámetros con valor por defecto (ej. `horas_extra=0`).
- Posicionales: deben ir en orden; `calcular_salario(50000, 3)`.
- Keyword arguments: `calcular_salario(base=50000, horas_extra=3)`.

- **Local:** Variables definidas dentro de la función solo existen dentro de ella.
- **Global:** Variables definidas fuera de cualquier función son accesibles dentro, pero no se recomienda modificarlas sin razón.
- **Mejor práctica:** Mantener funciones “puras”, evitando depender de variables globales.

Ejemplo: `def f(): x=10` — la variable `x` vive solo en `f()`.

Parámetros y Alcance ∈ Conflicto Global vs Local

```
1 x = 5  # global
2
3 def foo():
4     x = 10  # local
5     print("Dentro de foo, x =", x)
6
7 foo()
8 print("Fuera de foo, x =", x)
```

- Impresiones:
 - "Dentro de foo, x = 10"
 - "Fuera de foo, x = 5"
- **Evitar confusiones** usando nombres significativos y parámetros claros.

Módulos y Paquetes

Módulos y Paquetes ∈ ¿Qué es un Módulo en Python?

- Un **módulo** es un archivo **.py** que contiene definiciones de funciones, variables y clases.
- **Ventaja:** podemos importar este archivo desde otros scripts y reutilizar el código.
- **Ejemplo:** Crear un archivo **mifunciones.py** con varias funciones y luego:

```
import mifunciones
```

Módulos y Paquetes ∈ Ejemplo de Módulo: `mifunciones.py`

```
1  # mifunciones.py
2
3  def suma(a, b):
4      return a + b
5
6  def resta(a, b):
7      return a - b
```

Uso en otro archivo:

```
1  import mifunciones
2
3  print(mifunciones.suma(3, 4))
4  print(mifunciones.resta(10, 2))
```



```
1 from mifunciones import suma
2
3 resultado = suma(5, 7)
4 print(resultado)
```

- Trae solo la función **suma** del módulo.
- **Cuidado:** peligro de colisiones de nombre (si hay otra **suma**).
- **from mifunciones import *** trae todo, también puede causar conflictos.

- Un **paquete** es una carpeta que contiene un archivo `__init__.py` y varios módulos `.py`.
- Estructura típica:
- Para usarlo: `import mipaquete.modulo1`.

Uso: Organizar proyectos grandes en submódulos lógicamente separados.

Ejemplos y Actividad

Ejemplos y Actividad ∈ Ejemplo: Módulo con Funciones Físicas

```
1  # fisica.py (módulo)
2  import math
3
4  def energia_cinetica(m, v):
5      """Retorna la energía cinética:  $0.5*m*v^2$ """
6      return 0.5 * m * (v**2)
7
8  def energia_potencial(m, g, h):
9      """Retorna la energía potencial:  $m*g*h$ """
10     return m * g * h
11
12  def lanzar_proyectil(v0, ang):
13      """
14      Retorna el alcance teórico de un proyectil
15      en un tiro parabólico sin rozamiento.
16      """
17     rad = math.radians(ang)
18     g = 9.8
19     R = (v0**2 * math.sin(2*rad)) / g
20     return R
```

Ejemplos y Actividad ∈ Uso de `fisica.py` en un Script Principal

```
1  # main.py
2  import fisica
3
4  m = 10  # kg
5  v = 5   # m/s
6  ec = fisica.energia_cinetica(m, v)
7  print("Energía Cinética =", ec, "J")
8
9  distancia = fisica.lanzar_proyectil(20, 45)
10 print("Alcance estimado =", distancia, "m")
```

- **Objetivo:** Organizar funciones útiles en un módulo y probar su importación.
- **Instrucciones:**
 1. Crea un archivo `utilidades.py` con al menos 3 funciones (ej. `factorial(n)`, `es_primo(n)`, etc.).
 2. En un **notebook de Colab** o un script `main.py`, importa `utilidades` y prueba dichas funciones.
 3. (Opcional) Añade una cuarta función con un **parámetro por defecto** (`def hola_mundo(nombre="Mundo")`), etc.).

Sugerencia: Comenta tu código y explica la lógica de cada función.

- Formar **parejas o tríos**.
- Diseñar **un pequeño módulo** de funciones:
 - Matemáticas, Estadística básica, Conversión de unidades, etc.
- Utilizar dichas funciones en **otro script o notebook**.
- **Opcional:** Explorar la creación de una carpeta con `__init__.py` para armar un paquete simple.

- ¿Problemas al importar el módulo en Colab?
- ¿Cómo organizar los archivos en Google Drive?
- ¿Errores de `ModuleNotFoundError`?

Consulta en voz alta o pide ayuda a tus compañeros.

Ejemplos y Actividad ∈ Ejemplo de utilidades.py

```
1  # utilidades.py
2  def factorial(n):
3      """Calcula n! de manera recursiva."""
4      if n <= 1:
5          return 1
6      else:
7          return n * factorial(n-1)
8
9  def es_primo(num):
10     """Retorna True si 'num' es primo, False si no."""
11     if num < 2:
12         return False
13     for i in range(2, int(num**0.5)+1):
14         if num % i == 0:
15             return False
16     return True
17
18 def saludar(nombre="Mundo"):
19     """Imprime un saludo a 'nombre'."""
20     print(f"Hola, {nombre}!")
```

Ejemplos y Actividad ∈ Uso en un Notebook o Script

```
1 import utilidades
2
3 print("5! =", utilidades.factorial(5))
4 print("¿13 es primo?:", utilidades.es_primo(13))
5 utilidades.saludar()
6 utilidades.saludar("Física")
```

Resultado Esperado:

- 5! = 120
- ¿13 es primo?: True
- Hola, Mundo!
- Hola, Física!

- ¿Se entendió la **separación** entre la lógica (en un módulo .py) y el código principal?
- ¿Ventajas de tener todo en un solo archivo vs. múltiples módulos?
- ¿Dudas sobre parámetros y valores por defecto?

Conclusiones

- Apreciamos la **modularización** del código para mejor legibilidad y mantenimiento.
- Vimos **funciones**: sintaxis **def**, parámetros, **return**, docstring.
- Exploramos la creación de **módulos** y su importación en otros scripts o notebooks.
- Ahora estamos preparados para proyectos más grandes y ordenados.

- **Sesion 8 (Semana 4):** Continuaremos con el uso de módulos y paquetes, y profundizaremos en el uso de **pip** y librerías externas.
- **Tarea sugerida:**
 - Crear un pequeño proyecto con un paquete **mipaquete/** y varios módulos (ej. **mipaquete/calculos.py**, **mipaquete/utiles.py**, etc.).
 - Probar importarlos y usarlos en un script principal.

¡Gracias y hasta la próxima sesión!

- Guarda tus **notebooks** y archivos **.py** en Google Drive.
- Explora la documentación oficial de Python (sobre **funciones** y **módulos**).
- ¡Sigue practicando!