Programación para Física y Astronomía

Departamento de Física.

Corodinadora: C Loyola
Profesoras/es C Loyola / C Femenías / Y Navarrete / C Ruiz / F Bugini

Primer Semestre 2025

Universidad Andrés Bello
Departamento de Física y Astronomía

Universidad Andrés Bello®

python powered

Resumen - Semana 4, Sesión 1 (Sesión 7)

Repaso y Contexto

Funciones en Python

Parámetros y Alcance

Módulos y Paquetes

Ejemplos y Actividad

Conclusiones

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.
- **Objetivo de hoy**: Introducir **Funciones** en Python y exploración de **módulos/paquetes** con ejemplos prácticos.

Objetivos de la Sesión 7

- **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

¿Por qué usar Funciones?

- **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).

Sintaxis Básica de Funciones en Python

```python
def nombre_de_funcion(param1, param2, ...):
    """
    Documentación opcional (docstring).
    Explica qué hace la función, los parámetros y el retorno.
    """
    # bloque de código
    # opcionalmente retornar un valor
    return resultado
```

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

Ejemplo: Función para Calcular el Área de un Círculo

```python
import math

def area_circulo(radio):
    """
    Retorna el área de un círculo de radio 'radio'.
    Formula: pi * r^2
    """
    area = math.pi * (radio**2)
    return area

# Uso de la función
r = 5
a = area_circulo(r)
print("El área del círculo es:", a)
```

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

Parámetros y Alcance

Parámetros Posicionales y Opcionales

```python
def calcular_salario(base, horas_extra=0):
    """Calcula salario sumando base + 2000 por hora extra"""
    return base + 2000 * horas_extra

s1 = calcular_salario(50000)        # horas_extra por defecto: 0
s2 = calcular_salario(50000, 3)     # 3 horas extra
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)`.

Scope o Alcance de Variables

- **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()`.

Conflicto Global vs Local

```python
x = 5  # global

def foo():
    x = 10  # local
    print("Dentro de foo, x =", x)

foo()
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

¿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
```

Ejemplo de Módulo: `mifunciones.py`

```python
# mifunciones.py

def suma(a, b):
    return a + b

def resta(a, b):
    return a - b
```

Uso en otro archivo:

```python
import mifunciones

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

Importaciones Específicas

```python
from mifunciones import suma

resultado = suma(5, 7)
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.

Paquetes en Python

- 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

Ejemplo: Módulo con Funciones Físicas

```python
# fisica.py (módulo)
import math

def energia_cinetica(m, v):
    """Retorna la energía cinética: 0.5*m*v^2"""
    return 0.5 * m * (v**2)

def energia_potencial(m, g, h):
    """Retorna la energía potencial: m*g*h"""
    return m * g * h

def lanzar_proyectil(v0, ang):
    """
    Retorna el alcance teórico de un proyectil
    en un tiro parabólico sin rozamiento.
    """
    rad = math.radians(ang)
    g = 9.8
    R = (v0**2 * math.sin(2*rad)) / g
    return R
```

Uso de `fisica.py` en un Script Principal

```python
# main.py
import fisica

m = 10  # kg
v = 5   # m/s
ec = fisica.energia_cinetica(m, v)
print("Energía Cinética =", ec, "J")

distancia = fisica.lanzar_proyectil(20, 45)
print("Alcance estimado =", distancia, "m")
```

Actividad: Creando Tus Módulos

- **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.

Trabajo Colaborativo

- 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.

Espacio para Dudas y Apoyo

- ¿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.

Ejemplo de `utilidades.py`

```python
# utilidades.py
def factorial(n):
    """Calcula n! de manera recursiva."""
    if n <= 1:
        return 1
    else:
        return n * factorial(n-1)

def es_primo(num):
    """Retorna True si 'num' es primo, False si no."""
    if num < 2:
        return False
    for i in range(2, int(num**0.5)+1):
        if num % i == 0:
            return False
    return True

def saludar(nombre="Mundo"):
    """Imprime un saludo a 'nombre'."""
    print(f"Hola, {nombre}!")
```

Uso en un Notebook o Script

```python
import utilidades

print("5! =", utilidades.factorial(5))
print("¿13 es primo?:", utilidades.es_primo(13))
utilidades.saludar()
utilidades.saludar("Física")
```

Resultado Esperado:

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

Retroalimentación

- ¿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

Conclusiones de la Sesión 7

- 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.

Próximos Temas

- **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!