ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS
INTRODUCCIÓN AL DESARROLLO WEB
2025 II
LABORATORIO 21 – PYTHON: TÓPICOS AVANZADOS

I

OBJETIVOS

- Comprender la importancia de Python en el desarrollo web
- Comprender los tópicos avanzados de Python que son importantes para el desarrollo web
- Solucionar ejercicios aplicando los conocimientos obtenidos

TIEMPO ESTIMADO: 2 horas

II

CONSIDERACIONES DE EVALUACIÓN

- Se deberán utilizar los conocimientos impartidos en las clases teóricas
- Deberá utilizar nombre de variables significativos
- Deberá realizar pruebas adicionales
- El alumno deberá indicar en su código con quien colaboró, así sea la IA
- El alumno será requerido de realizar modificaciones en su código y responder a preguntas sobre el mismo
- Los ejercicios deberán realizarse en el laboratorio, a su ritmo y subirlos al aula virtual como AVANCE antes de finalizar la clase
- Todos los ejercicios completos, deberán ser subidos al aula virtual como TAREA, para lo cual se les dará un tiempo adecuado y deben cumplir el deadline estipulado. Esto se deberá cumplir, aunque en el laboratorio ya se hayan terminado todos los ejercicios
- El formato a usar para los avances y tareas será .zip, que contenga todos los archivos requeridos
- Utilizar Git con GitHub para el manejo de versiones de su trabajo, ocasionalmente se le solicitará que compartan sus repositorios

III

POLITICA DE COLABORACION

La política del curso es simple, a menos que se exprese lo contrario en el laboratorio, siéntase libre de colaborar con sus compañeros en todos los laboratorios, pero debe notificar expresamente con quien ha colaborado. La colaboración con alumnos, que no están matriculados en el curso está prohibida. Los laboratorios y asignaciones han sido desarrollados para ayudarlo a comprender el material. Conozca su código y esté preparado para revisiones individuales de código. Durante las revisiones es probable que se le pida realizar modificaciones y justificar sus decisiones de programación. Cada uno de sus ejercicios debe iniciar de la siguiente forma:

```
// Laboratorio  Nro x - Ejerciciox
// Autor: mi nombre
// Colaboró : el nombre
// Tiempo :
```

INDICACIONES GENERALES

a. En cada sesión de laboratorio, los ejercicios propuestos deberán ser guardados en la misma carpeta/directorio
b. La carpeta deberá tener el nombre del Laboratorio y el nombre del alumno, así por ejemplo:
Laboratorio 11 – Juan Perez
c. Utilice nombres significativos
d. Su código deberá estar correctamente indentado y preferentemente documentado
e. Deberá ser debidamente probado

MARCO TEORICO

1. Clases y Objetos

Una clase es un molde que define atributos y métodos. Un objeto es una instancia creada a partir de la clase. Permiten organizar código siguiendo el paradigma de Programación Orientada a Objetos (POO).

Conceptos clave:

Atributos: variables asociadas a un objeto
Métodos: funciones dentro de una clase
Constructor (__init__): función que inicializa atributos
Encapsulamiento: controlar acceso a atributos
Herencia: crear clases basadas en otras

Ejemplo:

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        return f"Hola, soy {self.nombre}"

p1 = Persona("Juancito", 25)
print(p1.saludar())
```

Ejemplo Herencia:

```
class Animal:
    def __init__(self, nombre):
        self.nombre = nombre

    def hablar(self):
        return "Sonido desconocido"

class Perro(Animal):
    def hablar(self):
        return "Guau!"

class Gato(Animal):
    def hablar(self):
        return "Miau!"

p = Perro("Firulais")
g = Gato("Michi")

print(p.nombre, p.hablar())  # Firulais Guau!
print(g.nombre, g.hablar())  # Michi Miau!
```

2. Manejo de Excepciones

Las excepciones permiten manejar errores en tiempo de ejecución sin detener el programa.

```
try:
      # Código que puede fallar
except TipoDeError:
      # Qué hacer cuando ocurre
else:
      # Se ejecuta si no hubo excepción
finally:
      # Se ejecuta siempre
```

Ejemplo:

```
try:
    x = int(input("Ingresa un número: "))
    print(10 / x)
except ZeroDivisionError:
    print("No se puede dividir entre 0.")
except ValueError:
    print("Debes ingresar un número válido.")
```

Ejemplo con raise:

```
def dividir(a, b):
    if b == 0:
        raise ZeroDivisionError("No se puede dividir entre cero.")
    return a / b

try:
    print(dividir(5, 0))
except ZeroDivisionError as e:
    print("Error:", e)
```

3. Programación Modular

Es dividir un programa grande en módulos (archivos .py) para mejorar organización, mantenimiento y reutilización.

Ventajas:

- Código más limpio y ordenado.
- Facilidad de mantenimiento.
- Reutilización en varios proyectos.

Ejemplo:

Archivo: operaciones.py

```
def sumar(a, b):

    return a + b

def restar(a, b):

    return a - b
```

Archivo principal:

```
import operaciones

print(operaciones.sumar(5, 3))
```

4. Archivos

Un archivo es un recurso externo (en disco) con el que tu programa puede interactuar. Python permite trabajar con archivos de texto (.txt), binarios (.jpg, .mp3), CSV, JSON, etc.

El ciclo típico es:

1) Abrir
2) Leer o escribir
3) Cerrar

Con with esto se hace automáticamente.

Modos comunes: (se pueden combinar)

- "r" → leer (error si no existe)
- "w" → escribir (crea o sobrescribe)
- "a" → agregar al final (append)
- "x" → crear (error si ya existe)
- "b" → modo binario (ej: "rb", "wb")
- "+" → lectura/escritura (ej: "r+")

Ejemplos:

```
f = open("datos.txt", "r")    # Abrir el archivo
contenido = f.read()          # Leer todo
print(contenido)
f.close()                     # Cerrar manualmente

f = open("notas.txt", "w")
f.write("Primera línea\n")
f.write("Segunda línea\n")
f.close()

with open("datos.txt", "w") as f:
    f.write("Hola mundo")

with open("datos.txt", "r") as f:
    print(f.read())

with open("datos.txt", "r") as f:
    for linea in f:
        print(linea)

with open("datos.txt", "r") as f:
    lineas = f.readlines()

with open("foto.jpg", "rb") as f:
    data = f.read()
```

5. JSON

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos. Python lo maneja con el módulo json.

Funciones más usadas:

- json.dumps(): Python → JSON (cadena)
- json.loads(): JSON (cadena) → Python
- json.dump(): Python → archivo JSON
- json.load(): archivo JSON → Python

Ejemplos:

```
import json

dict = {"nombre": "Ana", "edad": 30, "activo": True}
texto = json.dumps(dict)
print(texto)
```

```
import json

texto = '{"nombre": "Ana", "edad": 25, "cursos": ["Python", "Java"]}'

data = json.loads(texto)
print(data)
```

```
import json

with open("datos.json", "r") as f:
    data = json.load(f)
print(data)
```

```
import json

obj = {"nombre": "Luis", "edad": 30}

with open("datos.json", "w") as f:
    json.dump(obj, f, indent=4)
```

6. Concurrencia

La concurrencia permite ejecutar múltiples tareas *casi* al mismo tiempo. En Python se puede lograr con:

- Threads (módulo threading): tareas paralelas ligeras.
- asyncio: corrutinas y programación asíncrona.
- Procesos (módulo multiprocessing): verdaderamente paralelos.

Ejemplos

```
// hilos
import threading
import time

def descargar(nombre, tiempo):
    print(f"Iniciando {nombre}")
    time.sleep(tiempo)
    print(f"Finalizó {nombre}")

tareas = [
    ("Archivo A", 5),
    ("Archivo B", 5)
]

inicio = time.time()

hilos = []
for nombre, t in tareas:
    h = threading.Thread(target=descargar, args=(nombre, t))
    h.start()
    hilos.append(h)

for h in hilos:
    h.join()

print("Tiempo total:", time.time() - inicio)
```

```
// asyncio
import asyncio
import time

async def descargar(nombre, tiempo):
    print(f"Iniciando {nombre}")
    await asyncio.sleep(tiempo)
    print(f"Finalizó {nombre}")

tareas = [
    ("Archivo A", 5),
    ("Archivo B", 5)
]

async def main():
    inicio = time.time()

    await asyncio.gather(
        descargar("Archivo A", 5),
        descargar("Archivo B", 5)
    )

    print("Tiempo total:", time.time() - inicio)

asyncio.run(main())


// procesos
import multiprocessing
import time

def descargar(nombre, tiempo):
    print(f"Iniciando {nombre}")
    time.sleep(tiempo)
    print(f"Finalizó {nombre}")

tareas = [
    ("Archivo A", 5),
    ("Archivo B", 5)
]

if __name__ == "__main__":
    inicio = time.time()

    procesos = []
    for nombre, t in tareas:
        p = multiprocessing.Process(target=descargar, args=(nombre, t))
        p.start()
        procesos.append(p)

    for p in procesos:
        p.join()

    print("Tiempo total:", time.time() - inicio)
```

1. Crear un directorio que tenga su nombre y un subdirectorio laboratorio21. Recomendación: usar minúsculas, sin espacios, sin tildes ni “ñ” y con guiones medios o bajos
2. Utilizar los atajos de teclado o combinación de teclas para agilizar su trabajo
3. Usando una jerarquía de clases que permita calcular el área y el perímetro de Rectángulo, Triángulo y Círculo. Crear una lista con un objeto de cada figura y mostrar sus datos
4. Crea un programa que permita gestionar libros en una biblioteca usando clases y objetos.
   De los libros me interesa título, autor, año y ver si está disponible. Cada libro se debe poder prestar (si está disponible) y devolver (si no está disponible). Crear la clase biblioteca que administre los libros, debe manejar un conjunto de libros y se debe poder agregar libro nuevo, listar libros para mostrar todos los libros con su disponibilidad, buscar por autor y prestar libro (busca el libro por título y lo presta si está disponible). Crear una clase Libro Digital que herede de Libro y tenga propiedades adicionales de formato (ej: "PDF", "EPUB") y tamañoMB. Sobrescribir el método prestar para que siempre esté disponible (los libros digitales no se prestan físicamente).
   Crear al menos 3 libros físicos y 2 libros digitales, agregarlos a la biblioteca y probar los métodos:
   - Listar todos los libros
   - Prestar un libro físico
   - Prestar un libro digital 5 veces
   - Intentar prestar un libro ya prestado
   - Buscar libros por autor
5. Leer la operación como un solo string
   Ejemplo: "10 / 2"
   Separar los componentes y validar:
   - numero1 y numero2 deben poder convertirse a float
   - operador debe ser uno de: + - * /

   Realizar la operación, pero manejar las excepciones:
   - División entre cero → mostrar mensaje personalizado
   - Valores inválidos → manejar ValueError
   - Operador inválido → lanzar una excepción personalizada (crear clase que herede de Exception)
6. Crear un módulo geometria.py con lo hecho en el ejercicio 3 y utilizarlo desde otro archivo
7. Crear un programa que permita Copiar el contenido de un archivo a otro. 2 versiones: una para archivos de texto y otra para archivos binarios
8. Crear una lista de diccionarios donde cada diccionario represente un equipo de fútbol, con las siguientes claves:
   Nombre, país, nivelAtaque, nivelDefensa
   Luego, convierte la lista completa a una cadena JSON y muéstrala en pantalla con formato legible.
9. Simular el comportamiento de un programa que realiza 3 consultas a una base de datos remota, cada una con un tiempo de respuesta variable entre 1 y 5 segundos. Se quiere comparar la concurrencia usando hilos, tareas asíncronas y procesos
10. Crear un repositorio remoto en GitHub y subir tu repositorio local. Compartir URL y pdf con captura de pantalla del código de los archivos js y de la ejecución

I. TAREA PARA LA CASA: Complete todos los ejercicios.

Crear un documento docx/pdf con la solución de los ejercicios.

Subir el documento a la tarea **Tarea 21** del Aula Virtual respetando las fechas indicadas.