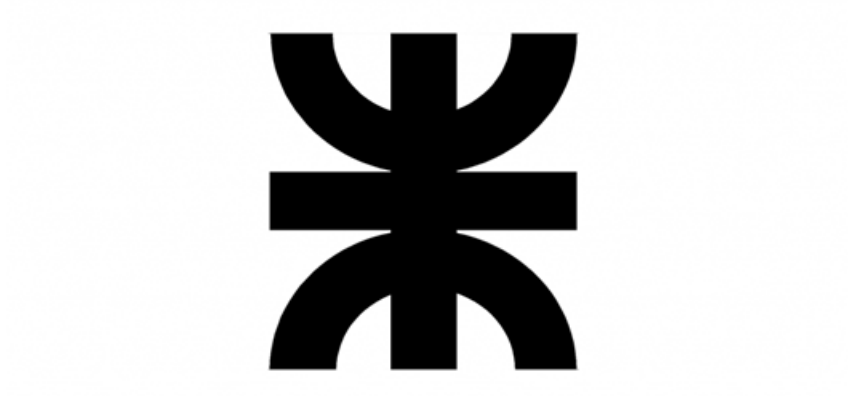

UNIVERSIDAD TECNOLÓGICA NACIONAL - FACULTAD REGIONAL CÓRDOBA

INGENIERÍA EN SISTEMAS DE INFORMACIÓN

INGENIERÍA Y CALIDAD DE SOFTWARE

PRÁCTICO 6 - TDD

Curso: 4k2

Docentes:

- Titular: Ing. Judith Meles.
- Jefa de Trabajos Prácticos: Ing. Cecilia Massano.
- Adscriptos:
    - Ezequiel Izaguirre.
    - Marcos Pomenich.

Grupo N°: 5

Integrantes:

- Andrade Patricio Valentin – 82770.
- Ayarzabal Fernandez Amilcar – 87661.
- Fasoletti Candelaria – 79708.
- Galiano Milagros – 89513.
- Gerbaldo Giuliana – 92773.
- Maldonado Franco – 92123.
- Nobile Bruno – 92098.
- Paredes Genaro – 85206.
- Torti Jeremías - 87531.
- Villane Ignacio – 62687.

Fecha de Entrega: 24/10/2025

Índice

Introducción

El presente documento tiene como objetivo establecer un conjunto de normas y convenciones de estilo para la escritura de código dentro del proyecto EcoHarmony Park. Estas directrices buscan promover la coherencia, legibilidad y claridad en el desarrollo, asegurando que todo el equipo de trabajo mantenga un estándar uniforme en la implementación del software.

En las siguientes secciones, se detallan las reglas adoptadas para el desarrollo en Python, utilizando como referencia la guía oficial PEP8, abarcando aspectos como nomenclatura de variables y clases, estructura del código, indentación, uso de espacios en blanco, documentación interna y pruebas automatizadas, entre otros elementos esenciales.

Al aplicar estas prácticas recomendadas, se busca alcanzar un código limpio, claro y mantenible, que refleje no solo la funcionalidad esperada del sistema, sino también un compromiso con la calidad y las buenas prácticas del desarrollo de software.

Reglas de estilo

Organización de carpetas

/frontend
- index.html
- styles.css
- script.js

/backend
- main.py
- parque_aventura.py
  /db
    - database.py

/tests
- test_comprar_entradas.py

run_tests.py

Convenciones de nombres

- Se usaron nombres claros y descriptivos para variables, funciones y clases.
- **Clases**: se utiliza la convención *PascalCase*.

```
class ConfiguracionParque:
```

- **Funciones y métodos**: se emplea *snake_case*.

```
def buscar_usuario_por_email(email):
```

- **Constantes**: se definen en *MAYÚSCULA_CON_GUINES_BAJOS*, indicando su carácter inmutable.

```
CANTIDAD_MAX_ENTRADAS = 10
```

- **Variables y atributos**: se nombran en *minúsculas_con_guiones_bajos*, con nombres descriptivos.

```
usuario_logueado = True
cantidad_entradas = 4
```

- **Métodos privados o internos**: se antepone un guión bajo (_) para indicar que no deben ser accedidos fuera de la clase.

```
def _validar_fecha(self, fecha):
```

- **Tests:** los nombres comienzan con la palabra *test* y son descriptivos, indicando de forma clara qué se valida.

```
def test_compra_fecha_pasada_falla(usuario_registrado):
```

Separación de responsabilidades

Cada archivo tiene una responsabilidad única:

- **parque_aventura.py**: lógica
- **ui_comprar_entradas.py**: ui
- **database.py**: base de datos
- **test_comprar_entradas.py**: tests

Formato del código

- **Indentación:** se utiliza la indentación de 4 espacios.

```
if usuario_valido:
    print("Compra realizada correctamente")
```

- **Espacios**: se agrega espacios alrededor de operadores y después de comas.

```
total = cantidad * precio_unitario
print("El total es:", total)
```

- **Importaciones**: las importaciones siguen un orden lógico: primero *bibliotecas estándar de Python*, segundo *bibliotecas externas* y por último *módulos internos del proyecto*.

```
import customtkinter as ctk
from datetime import date
from calendar import monthrange
from parque_aventura import ConfiguracionParque, Compra,
Usuario
```

Convenciones de clases y métodos

- **Métodos**: uso de *self* como primer argumento en métodos de instancia.

```
def mostrar_precio(self):
    return self.precio
```

- **Clases**: uso de *cls* como primer argumento en métodos de clase.

```
@classmethod
def incrementar_contador(cls):
    cls.contador += 1
```

Comentarios

- **Docstrings**: se emplean triple comillas " " " para documentar clases, métodos y funciones.

```
class Compra:
    """Clase que gestiona la compra de entradas."""
```

```
    def realizar_compra(self, usuario, cantidad, fecha):
        """Realiza la compra validando los datos ingresados."""
```

- **Comentarios breves**: se agregan comentarios para aportar información útil o explicaciones de decisiones.

```
# Validar que el usuario esté registrado antes de continuar
```

- **Bloques estructurados**: separa las secciones del código con líneas divisorias para mejorar la lectura.

```
# ——————————————————————————————————————
# MÉTODOS DE VALIDACIÓN
# ——————————————————————————————————————
```

Manejo de validaciones y errores

- **Validaciones centralizadas**: las comprobaciones de usuario, fecha y cantidad se realizan mediante métodos privados específicos dentro de la clase.

```
    def _validar_usuario(self, usuario):
        if not usuario.activo:
            self.errores.append("Usuario no activo")
```

- **Mensajes descriptivos**: los mensajes de error y confirmación son claros, breves y específicos.

```
    return {"ok": True, "mensaje": "Compra realizada correctamente"}
```

- **Validación de campos**: mensajes de error aparecen debajo del campo correspondiente, con clase *.error*

```
    <input type="date" name="fecha_visita" />
    <div id="error-fecha" class="error"> </div>
```

Bibliografía

- https://peps.python.org/pep-0008/#naming-conventions