BACKEND CON PYTHON

FUNDAMENTOS DE PROGRAMACIÓN

- Módulos
- Separación interfaz
- Buenas prácticas de programación

FUNDAMENTOS DE PROGRAMACIÓN

- Módulos
- Separación interfaz
- Buenas prácticas de programación

MÓDULOS

Cada vez que creamos un archivo .py, estamos creando un módulo Python.

El código que escribamos dentro de nuestro módulo, lo podremos reutilizar en otros módulos a futuro

- Facilita la reutilización de código y no tener que empezar siempre de cero.
- Podemos usar módulos escrito por nosotros mismos o por otros desarrolladores.
- Al reutilizar código de diversas fuentes, evitamos “reinventarnos la rueda”: si alguien ya escribió código con funcionalidades que nos sirven, no vale la pena volver a hacerlo desde ceros.
- Tener un proyecto con diferentes módulos nos permite organizar mejor las funcionalidades del programa y respetar conceptos fundamentales de la programación orientada a objetos como el encapsulamiento, del cual hablaremos más adelante.

MÓDULOS

Para crear módulos, simplemente debemos crear archivos .py, el código dentro de los módulos debe estar idealmente dentro de funciones, como lo discutiremos más adelante en las buenas prácticas de programación.

Vamos a tomar el siguiente ejemplo de un módulo llamado ejemplo1.py, que solo tiene una función que calculo el Índice Inventado del Curso, como lo indica el comentario: el dos veces la edad sobre la estatura al cuadrado.

ejemplo1.py

```
1  # -*- coding: utf-8 -*-
2
3  def calcular_IIC(estatura:float, edad:int)->str:
4      # El Indice se calcula como 2 veces la edad sobre la estatura al cuadrado
5      iic= (2*edad)/(estatura**2)
6      return round(iic,2)
```

MÓDULOS

Ahora, al interactuar con nuestro módulo, el usuario no tendrá mucha información:

Console 1/A

```
In [9]: runfile('C:/Users/[redacted]/Downloc
ejemplo1.py', wdir='C:/Users/[redacted]/Downl

In [10]: calcular_IIC(1.8,30)
Out[10]: 18.52
```

Por lo que vamos a crear un módulo intefaz1.py que use el módulo ejemplo1.py y le permite al usuario interactuar con el programa, dándole más información. Veremos un error en la línea 6 que arreglaremos en seguida:

interfaz1.py* | ejemplo1.py

```
1  # -*- coding: utf-8 -*-
2
3  nombre=input("Ingrese su nombre:")
4  estatura=float(input("Ingrese su estatura (en metros):"))
5  edad=int(input("Ingrese su edad:"))
6  iic=calcular_IIC(estatura,edad)
7
```

MÓDULOS

interfaz1.py* | ejemplo1.py

```
1  # -*- coding: utf-8 -*-
2
3  nombre=input("Ingrese su nombre:")
4  estatura=float(input("Ingrese su estatura (en metros):"))
5  edad=int(input("Ingrese su edad:"))
6  iic=calcular_IIC(estatura,edad)
7
```

El error se debe a que interfaz1 no puede usar la función calcular_IIC por qué está en otro módulo.

Para solucionar el error, debemos importar el módulo que contiene la función, en este caso es ejemplo1. Para ello, usaremos la instrucción import como se muestra a continuación en la línea 2:

interfaz1.py | ejemplo1.py

```
1  # -*- coding: utf-8 -*-
2  import ejemplo1
3
4  nombre=input("Ingrese su nombre:")
5  estatura=float(input("Ingrese su estatura (en metros):"))
6  edad=int(input("Ingrese su edad:"))
7  iic=ejemplo1.calcular_IIC(estatura,edad)
8  print("Hola, "+ nombre + ", su índice es: " + str(iic))
9
```

MÓDULOS

Al ejecutar el código anterior, podemos ver un resultado satisfactorio. Es importante resaltar que no tuvimos que volver a escribir el código de como calcular el IIC y redondearlo a 2 cifras. A continuación el resultado de la ejecución:

interfaz1.py | ejemplo1.py

```
1  # -*- coding: utf-8 -*-
2  import ejemplo1
3
4  nombre=input("Ingrese su nombre:")
5  estatura=float(input("Ingrese su estatura (en metros):"))
6  edad=int(input("Ingrese su edad:"))
7  iic=ejemplo1.calcular_IIC(estatura,edad)
8  print("Hola, "+ nombre + ", su índice es: " + str(iic))
9
```

Console 1/A

```
In [17]: runfile('C:/Users/[redacted]/Dow
interfaz1.py', wdir='C:/Users/[redacted]

Ingrese su nombre:Juan

Ingrese su estatura (en metros):1.8

Ingrese su edad:30
Hola, Juan, su índice es: 18.52
```

MÓDULOS

Hemos visto como crear y utilizar módulos propios, sin embargo, Python cuenta con librerías (conjunto de módulos) que tienen funciones implementadas de gran utilidad, además en internet podremos encontrar muchísimas más.

MÓDULOS

Una librería de Python que estaremos usando constantemente es math, que se importa igual que el ejemplo1 que ya vimos:

```python
# -*- coding: utf-8 -*-
import math

angulo = 45
seno_45 = math.sin(math.radians(angulo))
print("El seno de {0} grados es {1}".format(angulo,seno_45))

# Calcular el logaritmo natural de 2
logaritmo_2 = math.log(2)
print("El logaritmo natural de 2 es {0}".format(logaritmo_2))

# Imprimir el valor de pi
print("El valor de pi es {0}".format(math.pi))
```

La librería math cuenta con funciones trigonométricas, exponenciales y logarítmicas, de redondeo y constantes matemáticas como pi y euler.

```
In [26]: runfile('C:/Users/[redacted]/Downloads/Modu
[redacted].py', wdir='C:/Users/[redacted]/Downloads/M
El seno de 45 grados es 0.7071067811865476
El logaritmo natural de 2 es 0.6931471805599453
El valor de pi es 3.141592653589793
```

FUNDAMENTOS DE PROGRAMACIÓN

- Módulos
- Separación interfaz
- Buenas prácticas de programación

SEPARACIÓN INTERFAZ

En la primera clase vimos varios ejemplos de código y funciones Python que en algunos casos mezclaban operaciones de lógica (+, -, *, /, %, etc.) con funciones de Python que le permiten al usuario interactuar con la aplicación (input y print).

Aunque en un principio esto es normal, y dado que aprendimos a crear módulos, de ahora en adelante evitaremos mezclar la lógica con la interfaz (interacción usuario) por varias razones:

- Buena práctica de programación: ordenar el código y separarlo en módulos con funcionalidades específicas lo hace más fácil de modificar y mantener.
- Cuando trabajemos en proyectos siempre será así: backend (lógica) y frontend (interfaz).
- Para que el framework que aplicaremos para backend funcione correctamente, necesitamos separar lógica de interfaz, que es el patrón MVC del que hablaremos más adelante.

SEPARACIÓN INTERFAZ

Para los taller y proyecto de este primer módulo encontraremos módulos separados para la lógica y la interfaz.

En principio la interfaz será de consola y en los próximos módulos la reemplazaremos con una interfaz web.

Si no estuvieran separadas desde un principio, la transición de interfaz de consola a interfaz web sería muy difícil porque también tendríamos que modificar la lógica, pero si lo separamos correctamente, la transición será mucho más fácil.

SEPARACIÓN INTERFAZ

Retomemos el ejemplo del IIC como si tuviéramos la lógica e interfaz mezclada, que puede que funcione, pero no es lo ideal:

```python
# -*- coding: utf-8 -*-

def calcular_IIC()->None:
    nombre=input("Ingrese su nombre:")
    estatura=float(input("Ingrese su estatura (en metros):"))
    edad=int(input("Ingrese su edad:"))
    # El Indice se calcula como 2 veces la edad sobre la estatura al cuadrado
    iic= (2*edad)/(estatura**2)
    print("Hola, "+ nombre + ", su índice es: " + str(iic))
```

Si quisiéramos cambiar la interfaz de consola por una de escritorio o web, tendríamos que modificar la lógica, corriendo riesgos innecesarios de dañarla.

Pensemos que este programa solo tiene 6 líneas de código, pero: ¿cómo sería con 10000 o 1000000?

SEPARACIÓN INTERFAZ

Al separar interfaz y lógica

```python
# -*- coding: utf-8 -*-
import ejemplo1

nombre=input("Ingrese su nombre:")
estatura=float(input("Ingrese su estatura (en metros):"))
edad=int(input("Ingrese su edad:"))
iic=ejemplo1.calcular_IIC(estatura,edad)
print("Hola, "+ nombre + ", su índice es: " + str(iic))
```

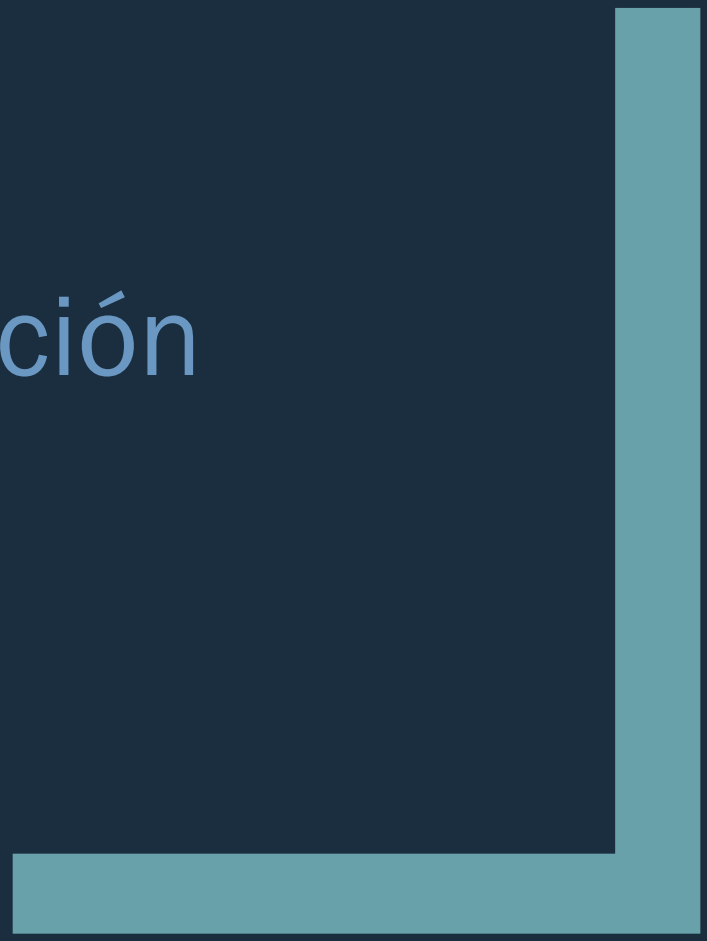
```python
# -*- coding: utf-8 -*-

def calcular_IIC(estatura:float, edad:int)->str:
    # El Indice se calcula como 2 veces la edad s
    iic= (2*edad)/(estatura**2)
    return round(iic,2)
```

Ahora podremos modificar la interfaz de consola por otra, sin correr el riesgo de alterar la funcionalidad de la lógica.

También si por alguna razón debemos cambiar las reglas de la lógica, por ejemplo, como calcular el IIC, solo tendremos que modificar el módulo de lógica sin alterar la interfaz.

FUNDAMENTOS DE PROGRAMACIÓN

- Módulos
- Separación interfaz
- Buenas prácticas de programación



BUENAS PRÁCTICAS

A continuación, mencionaremos algunas pautas, que si bien no son fundamentales para que los módulos o programas desarrollados funcionen correctamente, si son muy importantes para tener un código eficiente, limpio y mantenible.

Constantemente tendremos que hacer cambios y actualizaciones al código propio o de otros (por lo general el desarrollo de software se da en equipos de trabajo), por lo que es muy importante que sea muy fácil de entender el código, no solo por la persona que lo escribió, también para cualquiera que lo vaya a leer.

BUENAS PRÁCTICAS

Nombramiento de las variables y funciones:

- Poner nombres que den información de que es lo que están representando.
- Evitar poner nombres como “a”, “x”, en lugar “nombre”, “edad”.
- Evitar tener sinónimos en nombre de variables y funciones: por ejemplo, evitar tener “sumar”, “adicionar”, pero en caso de que sea estrictamente necesario, poner un comentario.

```
3  # No se entiende que representa x
4  x= 5
5
6  # Si se entiende que representa x
7  nota_maxima= 5
8
```

BUENAS PRÁCTICAS

Comentarios significativos:

- Poner comentarios dentro de las funciones, que expliquen que es lo que está haciendo el código.

```
3  def calcular_IIC(estatura:float, edad:int)->str:
4      # Bien, se entiende
5      # El Indice se calcula como 2 veces la edad sobre la estatura al cuadrado
6      iic= (2*edad)/(estatura**2)
7      return round(iic,2)
```

```
3  def calcular_IIC(estatura:float, edad:int)->str:
4      # Mal, no aporta información
5      # calcula icc
6      iic= (2*edad)/(estatura**2)
7      return round(iic,2)
```

BUENAS PRÁCTICAS

Signatura de las funciones:

- Poner nombres significativos para la función y parametros
- Especificar el tipo de datos de los parámetros
- Especificar el tipo de retorno

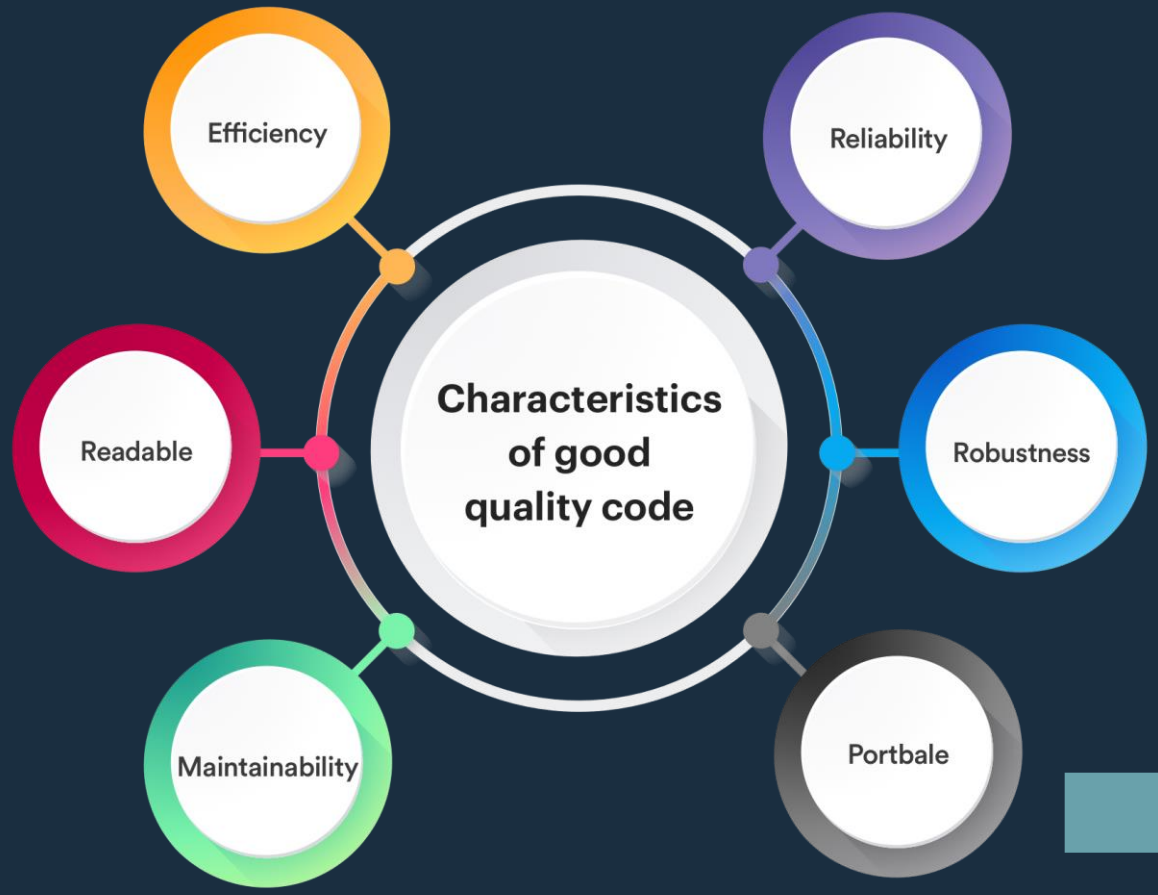
```
3  def calcular_IIC(estatura:float, edad:int)->float:
4      iic= (2*edad)/(estatura**2)
5      return round(iic,2)
```

Las dos hacen lo mismo, pero ¿cuál va a ser más fácil de leer, entender y modificar?

```
3  def func_1(a,b):
4      c= (2*a)/(b**2)
5      return round(c,2)
```

BUENAS PRÁCTICAS

Hay muchas buenas prácticas adicionales, que iremos viendo a medida que repasemos nuevos temas.



MANOS A LA OBRA

Taller 2 disponible en BloqueNeon