Nombre: Amanda Velasquez

Fecha: 1 de marzo de 2025

Curso: PT Full Stack Development with JavaScript, Python, React

M2C3 Python Assignment

I.Tipos de Datos en Python

1.1. Booleans

Solo tienen dos valores: True o False. Se usan para tomar decisiones en el código. Dependiendo de uno u otro se ejecuta algo o no. O se ejecuta algo distinto. En el siguiente ejemplo, se hará print del segundo.

```
edad = 18
tiene_DNI = False
if edad >= 18 and tiene_DNI:
  print("Puedes entrar")
elif edad >= 18 and not tiene_DNI:
  print("Eres mayor de edad, pero necesitas identificación")
else:
  print("No puedes entrar, eres menor de edad ")
```

1.2. Numbers

Los números se usan para contar o hacer cálculos. Dentro de estos hay tres tipos.

a. Enteros (Int). Son números naturales, sus inversos negativos y el cero. Ejemplo: 1, 2, -10, -3, etc.
b. Decimales (float). Son los números con punto decimal. Ejemplo: 2.3, 5.10, etc.
c. Complejos (complex). Números que tienen una parte imaginaria, como 2 + 3j (se usan en matemáticas avanzadas)

Un ejemplo de un cálculo sencillo en Python sería:
print(4+4) y esto dará 8 como resultado, así mismo se pueden hacer otras operaciones matemáticas.

1.3. Strings

El texto o cadenas de caracteres en Python se conoce como string y siempre va entre comillas dobles o simples. Además, con los strings se pueden hacer varias cosas, como concatenar, por ejemplo:

```
nombre = "María"
apellido = "Pérez"
saludo = "Hola,"
```

```
mensaje = saludo + " " + nombre + " " + apellido + "."
print(mensaje)
```

Esto imprime: Hola, María Pérez.

1.4. Bytes and byte arrays

Los bytes son como una forma especial de guardar información que las computadoras entienden mejor. En lugar de texto normal, los bytes usan código binario (ceros y unos). Se usan mucho para archivos, imágenes, videos o sonidos**,** porque la computadora no los entiende como texto, sino como datos en bruto.

Los bytes son como un tipo de texto, pero especial. Para escribirlos en Python, se agrega una b antes de las comillas: mensaje_bytes = b"Hola"

Un bytearray es como una lista de bytes. La diferencia con los bytes normales es que se pueden modificar. Ejemplo:

```
datos = bytearray(b"Hola")
print(datos)  La salida será  b'Hola'

Cambiamos la primera letra
datos[0] = 88
print(datos)   La salida será  b'Xola'
```

1.5. None

None es una forma de decir que una variable no tiene valor. Ejemplo:

```
resultado = None
if resultado is None:
   print("Todavía no hay resultado")
```

1.6. Lists

Una lista es como una caja donde se pueden guardar muchas cosas juntas y cambiarlas cuando se quiera. Estas van entre corchetes. Se pueden extraer cosas de la lista indicando su posición. También se pueden agregar cosas a la lista con .append(), por ejemplo:

colores = ["rojo", "azul", "verde"]
colores.append("amarillo") Ahora la lista incluirá "amarillo"

1.7. Tuples

Una tupla es como una lista, pero una vez que se crea, no se puede modificar. Se usan para datos constantes como coordenadas o datos fijos. Además, van entre paréntesis. Ejemplo:
dias_semana = ("Lunes", "Martes", "Miércoles", "Jueves", "Viernes")

1.8. Sets

Un set o conjunto es como una bolsa donde se meten cosas sin repetirlas. No tienen orden, por lo que no se puede acceder a sus elementos por índice, pues daría error. Estos van entre llaves { } Ejemplo:
amigos = {"Carlos", "Sofía", "María"}

1.9. Dictionaries

Un dict (diccionario) es como una agenda de contactos: cada cosa tiene un nombre (clave) y un valor. Ejemplo:

```
persona = {
  "nombre": "Carlos",
  "edad": 30,
  "ciudad": "Madrid"
}
```

Yara obtener un valor, se usa la clave:
persona = {
print(persona["nombre"]) Nos dará el nombre de Carlos

También podemos modificar los valores así:
persona["edad"] = 31
print(persona["edad"]) Ahora nos dirá 31

II. Convención para la nomenclatura de variables en Python

En Python, se recomienda seguir la convención de nomenclatura de PEP 8, que es la guía oficial de estilo para escribir código limpio y entendible, listadas a continuación.

2.1. Usar snake_case (minúsculas con guiones bajos)

La forma correcta de nombrar variables en Python es usando letras minúsculas y separando palabras con un guion bajo (_).

Correcto:	Incorrecto:
nombre_completo = "Carlos Pérez"	NombreCompleto = "Carlos Pérez"

2.2. No usar espacios ni caracteres especiales

Los espacios no están permitidos en los nombres de variables. Para separar palabras se usa _ (guion bajo).

Incorrecto:	Correcto:
nombre usuario = "Ana"	nombre_usuario = "Ana"
nombre-usuario = "Ana"	

2.3. No comenzar con un número

Los nombres de variables no pueden empezar con un número, pero pueden contener números después de la primera letra.

Incorrecto:	Correcto:
2nombre = "Pedro"	nombre2 = "Pedro"

2.4. Evitar palabras clave de Python

No usar palabras reservadas de Python (if, while, return, def, etc.).

Incorrecto:	Correcto:
def = "Función"	mi_definicion = "Función"
print = "Hola"	mensaje = "Hola"

2.5. Ser descriptivo y claro

Usa nombres que expliquen claramente qué almacena la variable. Un buen nombre de variable evita la necesidad de comentarios.

Incorrecto:
x = 25
y = "Carlos"

Correcto:
edad_usuario = 25
nombre_cliente = "Carlos"

2.6. Usar mayúsculas solo para constantes

En Python, las constantes (valores que no deben cambiar) se escriben en mayúsculas con _.

Correcto:
PI = 3.1416

Incorrecto:
pi = 3.1416

2.7. Usar nombres cortos pero significativos

No usar nombres demasiado largos o demasiado cortos.

Incorrecto:
a = 100 (no se entiende qué significa)
edad_del_usuario_que_se_registro_en
_2023 = 25

Correcto:
edad_usuario = 25

III. Heredoc en Python

Heredoc se usa para describir cadenas de texto multilínea que se escriben de manera clara y legible.
Se usan triple comillas (""" o ''') para definir cadenas de varias líneas sin necesidad de concatenarlas manualmente con \n.
Ejemplo:

```
mensaje = """Nullam id dolor id nibh ultricies vehicula ut id elit.
Nullam quis risus eget urna mollis ornare vel eu leo.

Vestibulum id ligula porta felis euismod semper. Cum sociis natoque penatibus et magnis
dis parturient montes, nascetur ridiculus mus. Cras justo odio, dapibus ac facilisis in.

Integer posuere erat a ante venenatis dapibus posuere velit aliquet.
"""
```

IV. Interpolación de cadenas

La interpolación de cadenas permite insertar valores dentro de una cadena de texto de manera sencilla. En vez de concatenar (+), se usa una plantilla con espacios reservados y se reemplazan esos espacios con valores. Se usa f-strings. Por ejemplo:

```
nombre = "Ana"
edad = 25

mensaje = f"Hola, me llamo {nombre} y tengo {edad} años."
print(mensaje)
```

También permite hacer cálculos y funciones dentro de la interpolación:

```
precio = 15.5
cantidad = 3
total = f"El total a pagar es: {precio * cantidad} €."
print(total)
```

V. Uso de comentarios en Python

Mientras el código puede cambiar con el tiempo, los comentarios suelen quedarse atrás, lo que genera confusión en lugar de claridad. Un comentario que antes era útil puede volverse obsoleto, dando instrucciones incorrectas y generando errores en el futuro.

En lugar de depender de comentarios para describir el comportamiento del código, un enfoque más efectivo es nombrar correctamente las variables, funciones y clases. Si un método tiene un nombre claro y preciso, no necesita un comentario para explicar lo que hace. Sin embargo, el código puede ser beneficioso para la organización del código. Por ejemplo: en archivos extensos, como hojas de estilo CSS, los comentarios pueden ayudar a dividir secciones sin describir comportamientos.

```
/* Estilos del encabezado de navegación */
.navbar { ... }
```

VI. Diferencias entre aplicaciones monolíticas y de microservicios

La principal diferencia es que las aplicaciones monolíticas tienen todo en una sola estructura, mientras que las de microservicios están divididas en varias estructuras separadas.

6.1. Monolítico = Todo Junto

En una aplicación monolítica, todo el código está en un solo proyecto y se ejecuta como una única unidad. Por ejemplo: Una tienda en línea. En una aplicación monolítica, todo estaría dentro del mismo código y servidor:

a. Módulo de usuarios (registro, login)
b. Módulo de productos (catálogo, stock)
c. Módulo de pagos (procesar compras)
d. Módulo de envíos (seguimiento de paquetes)

Todos estos módulos comparten la misma base de datos, código y servidor. La ventaja es que es más simple de construir y más rápido. La desventaja es que si algo falla (por ejemplo, el sistema de pagos), puede afectar toda la aplicación.

6.2. Microservicios = Todo Separado

En una aplicación de microservicios, en lugar de tener todo en una sola estructura, cada módulo es un servicio independiente que se comunica con los demás. Entonces, en una tienda en línea, por ejemplo, iría así:

a. Un microservicio solo para usuarios.
b. Un microservicio solo para productos.
c. Un microservicio solo para pagos.
d. Un microservicio solo para envíos.

Cada microservicio tiene su propio código, base de datos y puede estar en servidores distintos. La ventaja es que, por ejemplo, si falla el sistema de pagos, la tienda sigue funcionando y los usuarios pueden seguir navegando. La desventaja sería que es más complejo de desarrollar y requiere más esfuerzo para que los servicios se comuniquen correctamente.