M2C3 – Python – Preguntas Teóricas

¿Cuáles son los 4 tipos de datos en Python?

Los 4 tipos de datos en Python son:

- **Números**, por ejemplo: **number = 2**
Se incluyen dentro de los números los enteros, con decimales, fracciones, etc.

-**Strings**(cadenas), por ejemplo: **name = "Ana"**
Una cadena puede contener uno o varios caracteres, espacios, números y otros tipos de caracteres.

- **Booleans**(booleanos), sólo hacen referencia a dos valores, verdadero o falso. Ejemplo: **answer = True** (o False).

- **Lists**, que tienen una sintaxis similar a los arrays(arreglos) en JavaScript, por ejemplo: **list = ['1','2','3']**
Una lista puede contener uno varios elementos dentro y pueden contener diferentes tipos de datos.

¿Qué tipo de convención de nomenclatura debemos usar para las variables en Python?

Se debe usar **snake_case**. La variable se declara en minúsculas. En caso de que sean dos palabras se separan por un guion bajo y ambas se escriben en minúsculas. Sin embargo, también puede usarse **camelCase** (también llamada lowerCamelCase) o **PascalCase** (también conocida como UpperCamelCase).

Las variables no pueden comenzar con números, pero pueden contenerlos. Deben evitarse caracteres confusos, en especial cuando se trate de variables con nombres de un único caracter como L minúscula, i mayúscula, o mayúscula y 0 (cero).

Los nombres de las variables deben ser elegidos de tal modo que sea fácil saber a qué hacen referencia. De esa forma, si alguien hereda nuestro código o si debemos que retomarlo luego, será más fácil saber a qué refiere cada cosa, y eso nos permitirá trabajar de forma más rápida y eficiente, sin necesidad de comentarios prescindibles.

¿Qué es un heredoc de Python?

Un heredoc es string multi-línea. Se señala con ” ” ”cuando inicia y finaliza el string.
Si fuera necesario señalar los saltos de línea, podría hacerse utilizando \n.

¿Qué es la interpolación de cadenas?

La interpolación de cadenas nos permite ejecutar código de Python dentro de un string. Ejemplo:

```
nombre = 'Ana'

saludo = f'Hola {nombre}'

print(saludo)
```

Podría también aplicarse con un comentario multi-línea, utilizando ” ” ”al inicio y cierre del string.
Ejemplo:

```
nombre = 'Ana'
```

```
curso_py = 'Python Avanzado'

saludo = f"""¡Hola {nombre}!

Gracias por comprar nuestro curso {curso_py}.

Un saludo,

Jordan"""

print(saludo)
```

¿Cuándo debemos usar comentarios en Python?

Lo ideal es usar el mínimo de comentarios que sea posible. Para ello daremos a los elementos, clases, variables, etc, nombres que sean claros y "autoexplicativos".

Un código claro y fácil de interpretar hará que los comentarios sean innecesarios en la mayoría de los casos, y además nos evitará ciertos problemas como que olvidemos borrar o actualizar los comentarios si modificamos nuestro código.

¿Cuáles son las diferencias entre las aplicaciones monolíticas y de microservicios?

Las aplicaciones monolíticas se construyen como una sola unidad, existe un único código que lo incluye todo, desde la interfaz de usuario hasta la lógica de negocios.

En estas aplicaciones todo se encuentra contenido en un mismo servidor y las diferentes partes que la componen se encuentran relacionadas unas con otras.

Estas aplicaciones tienen como **ventajas** que son aptas para desarrolladores con poca experiencia, son más rápidas porque no requieren comunicarse vía API, son menos costosas de desarrollar porque requieren menos recursos.

La principal **desventaja** de estas aplicaciones es que si algo falla, al estar todo interrelacionado, se produce un efecto dominó y la aplicación entera deja de funcionar.

Las aplicaciones de microservicios se dividen en diversos servicios que se encargan de una o varias funcionalidades. Esos servicios se desarrollan y almacenan de manera independiente y se comunican por mensajería o protocolos "https".

En estas aplicaciones, no hace falta que los servicios sean almacenados en el mismo servidor, ni que accedan a los mismos archivos.

Ventajas de este tipo de arquitectura:

- Los diferentes servicios pueden desarrollarse en remoto y de manera independiente.

- Si ocurre un fallo que afecta parte de la aplicación, no se da el efecto dominó, por lo que esa funcionalidad de la aplicación deja de estar disponible, pero el resto se mantienen operativas.

-En caso de decidir escalar la aplicación, no hace falta modificar todo sino la funcionalidad que se desea mejorar, por lo que hay menos probabilidades de que la aplicación falle.

Desventajas:

- No es ideal para desarrolladores sin experiencia.

- Requiere más recursos por lo que es más cara.

- Tarda más, ya que las diferentes partes de la aplicación deben comunicarse unas con otras.

M2C3 – Python Assignment

Exercise 1: Create a string, number, list, and boolean, each stored in their own variable.

String	Number	List	Boolean
name = 'Juana Suarez'	number = 5	lista = ['1','2','3','4']	booleano = False

Exercise 2: Use an index to grab the first 3 letters in your string, store that in a variable.

```
name = 'Juana Suarez'
print(name[0:3])
name_two = name[0:3]
print(name_two)
```

Exercise 3: Use an index to grab the first element from your list.

```
lista = ['1','2','3','4']
print(lista[0])
```

Exercise 4: Create a new number variable that adds 10 to your original number.

```
number = 5
new_number = number + 10
print(new_number)
```

Exercise 5: Use an index to get the last element in your list.

```
lista = ['1','2','3','4']
print(lista[-1])
```

Exercise 6: Use split to transform the following string into a list.

```
names = 'harry,alex,susie,jared,gail,conner'

print(names.split(','))
```

Exercise 7: Get the first word from your string using indexes. Use the upper function to transform the letters into uppercase. Create a new string that takes the uppercase word and the rest of the original string.

```
name = 'Juana Suarez'
print(name)

string_indice = name[0:5]
string_mayus = string_indice.upper()
print(string_mayus)

string_final = name.lstrip("Juana")
print(string_final)

new_string = string_mayus + string_final
print(new_string)
```

--> Este fue el ejercicio que más me costó porque no se me ocurría cómo hacerlo. Primero intenté con partition pero no cumplía con la condición del index y luego con l.strip. He revisado mis apuntes y he visto que la forma más sencilla de hacerlo era usar negative index.

```
name = 'Juana Suarez'
print(name)

upper_variable = name[0:4].upper()
print(upper_variable)

new_variable = upper_variable + name[-8:]
print(new_variable)
```

Exercise 8: Use string interpolation to print out a sentence that contains your number variable.

```
number = 5

sentence = f"El día {number} de marzo de 1933, el Partido Nazi gana las elecciones alemanas"

print(sentence)
```

Exercise 9: Print “hello world”.

```
mensaje = ' "Hello World" '
print(mensaje)
```