Checkpoint3

1. **Tipos de datos**:

Hay nueve tipos principales de datos que se usan en Python. Cada uno de los datos que guardas en una variable, Python los deduce automáticamente.

-Booleans. Es un tipo de dato que siempre nos va a dar verdadero o falso.

Si tu pones 2>3 nos va a dar un resultado de falso. Sólo almacena dos datos o verdadero o falso. Le das una información a tu ordenador en tu aplicación para que compruebe si el contenido que mandas es correcto o no.

Si quieres saber si un artículo está publicado o no. Si se almacena en un booleans estará en falso hasta que dicho artículo esté publicado.

-Numbers

Python es de los mejores lenguajes de programación que trabaja con números.

Tenemos tres tipos de datos numéricos: enteros, flotantes o fracciones.

Lon números enteros son cualquier número exacto ,1 4, 5666…

Los números flotantes van a ser los decímales: 0,5, 7,999…

También pueden ser fracciones: 1/7..

-Strings-

Son tipos de datos de cadenas. Pueden ser de cualquier tipo de secuencia. Algunos muy específicos como “name”(tienes claro que nombrará un nombre), también puede ser un documento entero, todo tipo de formas donde se pueda integrar una cadena de texto(una oración por ejemplo sentence = “this dog is white”).

Siempre aparecerán envueltas en comillas, que pueden ser simples o dobles. El lenguaje Python las va a entender igual. Siempre detrás del nombre de la cadena el signo = y después comillas antes y después del contenido de la cadena.

Una cadena no puede combinarse con un número.

-bytes and byte arrays

Se utilizan en desarrollo avanzado a más alto nivel.

-None.

En la gran mayoría de lenguajes dinámicos siempre existe el concepto de “ninguno” o “nulo”.

Un ejemplo claro es cuándo quieres establecer una variable, pero no tienes claro que valor asignar a algo dentro de ella. Aquí está bien utilizar este tipo de datos “none”. Luego cuando lo tienes claro reordenas y ya no quieres utilizar “none” sino que quieres asignarle otro contenido.

-Lists, Tuples, Sets, dictionaries.

Son todos estructuras de datos. Son uno de los bloques de construcción más grandes cuando se trata de construir aplicaciones web.

2. Momenclatura para usar variables.

Los nombres de las variables deben escribirse por norma general en minúsculas. Tienen que ser letras o números y siempre separadas las palabras por un guion bajo. Nunca deben empezar por un número.

A este tipo de estilo para trabajar en Python se denomina snake_case. Es decir, utilizar generalmente todo en minúsculas y utilizar el guion bajo para separar palabras. Los espacios no existen porque quedan subrayados por el guion bajo y las primeras letras de cada palabra en minúscula.

Nunca se debe usar L minúscula, la letra o (ni mayúscula ni minúscula) ni la i. La L minúscula y la i mayúscula pueden parecer iguales y la o puede confundirse con un cero.

Después de asignar un nombre a una variable se utiliza = y lo que escribas después del signo igual es lo que va a contener dentro la variable.

sentence_two = “this is a cat”

Las variables deberían ser lo más descriptivas posibles

name = “” nos indica perfectamente que va a almacenar un nombre. La variable se explica a sí misma.

Cuanto más descriptivas sean las variables menos comentarios deberemos hacer.

3.**Heredoc.**

Heredoc es un formato que se usa cuando trabajamos con cadenas de múltiples líneas. Si tienes varias líneas o párrafos en una cadena generalmente en programación se denomina heredoc.

Si tenemos tres párrafos la manera de que Python entienda que es una sola cadena y los párrafos van en la misma cadena es poniendo “”” al principio y “”” al final del texto.

#heredoc

Content = “””

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.

“””

print(content)

Si imprimimos esto veremos que el heredoc consigue enviar los párrafos de forma correcta.

El heredoc conecta todos los caracteres de una nueva línea.

Una cadena normal no funciona para múltiples líneas de ahí la necesidad de usar heredocs.

Si envolvemos cualquier texto en un heredoc Python hará la tarea de agregar espacios y el ordenador nos dará el resultado.

4.**Interpolacikón de cadenas.**

Sirve para darte la habilidad de ejecutar el código Python dentro de las cadenas. En cualquier aplicación es raro que el código sea cerrado, que sea estático, es decir que no puedas cambiar las cadenas. Algunos datos pueden estar en un código cerrado pero otros necesitan ser dinámicos (que tengas la posibilidad de añadir o cambiar cosas en los elementos). Esto va a depender de como lo utilice el usuario. Esto nos lo va a permitir la interpolación de cadenas.

La forma de utilizar es con f (bandera de formato) comillas lo que quieras poner y luego entre corchetes {} lo que esté aquí va a ser tratado como código de Python y lo va a interpretar porque después de ver la f busca los corchetes para ver lo que tiene que ejecutar.

name = “Alba”

greeting = f’hello {name}’

print(greeting)

Nos va a dar ‘hello Alba’. Cualquier otra cosa que incluyas en los corchetes lo va a ejecutar. Si ponemos {5 + 5} al dar print(greeting) nos va a decir ‘hello 10’. Los corchetes le indican a python que se va a usar una interpolación de cadenas.

Con la interpolación de cadenas podemos cambiar un mensaje que vaya dirigido específicamente a un usuario, cambiando lo que incluyamos dentro de los corchetes. Esto es lo que hace dinámico al lenguaje.

```
name = "Alba"

product = "white jeans"

content = f"""

hi {name}


thanks for using our {product}

"""

print(content)
```

Esto nos diría “hi Alba thanks for using our white jeans”.

Lo que te permite es que puedas cambiar el nombre de usuario y el producto cuando lo necesites , por eso es dinámico y se necesita la interpolación de cadenas.

5. **Los comentarios en Python.**

Sirven para describir la intención que vas a tener con el código. Para organizar lo que vas a desarrollar. Son buenos si tienes un archivo grande y quieres tenerlo ordenado.

Sirven como una nota para dar una explicación sobre el código fuente, sobre lo que vamos a programar. Pueden servir para darnos documentación. También sirven para que sea más fácil entender el programa. Además, los comentarios facilitan que el programador recuerde las cosas complejas agregadas al código y ayudar los componentes más importantes de un programa.

Hay tres clases de comentarios:

-los comentarios de una sola línea

Comienzan con el carácter # al principio y van hasta el final de la línea.

esto es una prueba. Si imprimes esto en la terminal te va a dar tres …, si quieres seguir con tu código tienes que volver a dar return.

```
#esto es una prueba

…

>>>
```

(ignora el comentario)

-los comentarios en línea

name = “Alba González” # TODO: Split en dos variables

TODO así en mayúsculas. Lo puedes encontrar en casi todos los lenguajes de programación, es como un símbolo universal. Significa que colocas ese comentario porque vas a volver a trabajar en este elemento de tu programa y vas a cambiar cualquier cosa más adelante.

Es solo un punto de referencia, este comentario será ignorado en el código.

print(name) Alba González

-los comentarios de varias líneas, multilínea.

Se usan cuando tienes más de una línea o párrafos.

"""

Multi-Line:

Aenean lacinia bibendum nulla sed consectetur.

Aenean lacinia bibendum nulla sed consectetur.

Aenean lacinia bibendum nulla sed consectetur.

"""

Tres comillas dobles al principio de lo que quieras en la parte superior comentar y tres comillas dobles después de la línea inferior.

Una cosa perjudicial de los comentarios es que si no los actualizas se vuelven inútiles.

Otra clave para no tener que usar muchos comentarios es elegir perfectamente el nombre de clase o de variable. Si el nombre describe bien lo que va a estar dentro de ese trozo de código, entonces los comentarios no son tan necesarios.

6. **Aplicaciones monolíticas y de microservicios.**

La aplicación monolítica es una aplicación basada en los sistemas tradicionales. Todo se encuentra junto y se comunican en la misma base de datos y en el mismo servidor. Es un mismo sistema de archivos que se comunican entre sí.

La aplicación monolítica es más sencilla, más rápido su desarrollo, no se tienen que comunicar a través de Api.

Pero hacer un sólo cambio en alguno de sus elementos puede perjudicar a toda la aplicación. Puede crear un efecto dominó y causar problemas en diferentes partes de una base del código.

En las aplicaciones de microservicios, en cambio, esto puede ser un beneficio. Puedes ampliar componentes en cualquier lugar sin necesidad de cambiar o ampliar la aplicación entera.

Se puede aislar cualquier problema de forma rápida y fácil. Si un elemento nos falla no tiene porqué fallar todo el sistema.

Las aplicaciones de microservicios son una arquitectura en la que cada característica es su propia aplicación. Cada archivo tiene su propio servidor separado del resto y con una aplicación única. Las API se conectan entre sí pero no dependen unos de otros.

Para aplicaciones pequeñas es mejor utilizar las monolíticas porque no requieren mucho trabajo para construirse y no son necesarios muchos recursos.

Si tienes un trabajo grande y puedes dividir el trabajo en secciones y para diferentes personas aquí usarías las aplicaciones de microservicios. Cada persona puede hacer una parte del sistema y realizar su trabajo sin preocuparse de nada más de la aplicación.