**Python intermedio:**

**Zen de Python:** Python es un lenguaje conito, limpio y estético creado en 1999. Los principios de Python son para escribir un código de manera clara, sencilla y precisa.

* **Bello es mejor que feo**:  
  Python es estéticamente superior a cualquier otro lenguaje de programación. Al momento de escribir código, es mejor que sea de manera limpia y estética.
* **Explícito es mejor que implícito**:  
  Hacer más fácil que las otras personas entiendan el código.
* **Simple es mejor que complejo**:  
  Es mejor tener una implementación simple, que ocupe pocas líneas de código y sea entendible, a que sea una larga y complicada.
* **Complejo es mejor que complicado**:  
  Es mejor extenderse y hacer más complejo el código para que el código si se entienda, a que hacerlo simple y mal.
* **Plano es mejor que anidado**:  
  El *anidamiento* es cuando tenemos un bloque de código que depende de otro bloque de código. Esto se nota en Python por la indentación, nos quedarían estos bloques muy corridos a la derecha.  
  Es mejor evitar el anidamiento, y hacer las cosas de **manera plana**.
* **Espaciado es mejor que denso**:  
  Por la indentación de Python (sus sangrías), este principio se nos hace imposible de esquivar. El código inevitablemente es espaciado.
* **La legibilidad es importante**:  
  Es importante que otros programadores puedan entender lo que estamos escribiendo. Esto hace más fáciles las cosas cuando trabajemos con otros en los proyectos.
* **Los casos especiales no son lo suficientemente especiales como para romper las reglas, sin embargo, la practicidad le gana a la pureza**:  
  Siempre que podamos respetar estas reglas que nos plantea Python, es mejor así. Sin embargo, si por el hecho de hacer un código muy puro o muy ‘Pythonico’, este pierde legibilidad, es mejor ser más prácticos y romper o saltearnos algunas de estas reglas para que el código sea más eficiente. Por lo tanto, llegado el momento debemos decidir si es mejor hacer las cosas de manera pura o práctica.
* **Los errores nunca deberían pasar silenciosamente (a menos que se silencien explícitamente)**:  
  Manejar los errores es fundamental. Cada error nos dice algo y hay que prestarle atención. A menos que seas capaz de silenciar un error explícitamente, aunque para esto hay que tener criterio.
* **Frente a la ambigüedad, evitar la tentación de adivinar**:  
  Nuestro código debería solamente tener una interpretación. Si en un contexto significa algo, y en otro otra cosa, es mejor que lo revisemos y busquemos una solución.
* **Debería haber una, y preferiblemente sola una manera obvia de hacerlo. (A pesar de que esa manera no sea obvia a menos que seas holandés)**:  
  Esto hace referencia al creador de Python ''Guido van Rossum", que de manera muy inteligente encontraba las soluciones precisas a los problemas, y deberíamos imitarlo.
* **Ahora es mejor que nunca**:  
  Es mejor desarrollar nuestra solución cuánto antes, no dejarlo para mañana o para más adelante.
* **A pesar de que nunca es muchas veces mejor que *ahora* mismo**:  
  Si por hacer las cosas ya y tenemos poco tiempo, si es mejor dejarlo para después y no hacerlo apurado y mal.
* **Si la implementación es difícil de explicar, es una mala idea, y si es fácil de explicar, es una buena idea**:  
  Si somos capaces de explicar nuestra implementación a otros desarrolladores paso a paso, es una buena idea. En cambio, si no podemos hacerlo, significa que ni nosotros entendemos la implementación y deberíamos repensar nuestra forma de encarar la solución.
* \*\*Los espacios de nombres son una gran idea, ¡Tengamos más de esos! (namespaces):  
  Es el nombre que se ha indicado luego de la palabra import, es decir la ruta (namespace) del módulo. (Lo veremos a profundidad más adelante).

Import this 🡪 para abrir el zen de Python en la consola de Python.

**Documentación:**

Información que explica como funciona el lenguaje, las piezas que hacen que todo funciona como tiene que hacerlo.

Índice Pep: propuestas de mejora de Python, son los que nos dicen como se debería escribir el lenguaje de manera correcta y como funciona.

Resumen Pep 8:

**Entorno virtual:**

Un módulo es código escrito por otra persona que funciona para solucionar un problema de manera rápida, de esta forma se evitar repetir soluciones, un ejemplo de esto es el módulo random. El entorno virtual permite que cada proyecto tenga una versión de Python con sus propios módulos, de esta forma se evitan conflictos y que, al momento de actualizar los módulos, se pueda elegir en que proyectos se actualiza y en cuales no:

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamenteImagen que contiene Diagrama

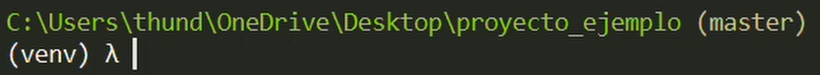
Descripción generada automáticamente

**Creación de un entorno virtual:**

Para crear un módulo desde la consola Cmder hay que crear una carpeta (mkdir proyecto\_ejemplo), y entramos en la carpeta (cd proyecto\_ejemplo), ahora vamos a crear un repositorio de git (git init) dentro de la carpeta, y para creamos el entorno virtual o sea un Python nuevo para ese proyecto (py -m venv venv).

(- dice al comando que se va a modificar el funcionamiento normal, -m es para llamar un módulo interno del lenguaje)

Que el entorno virtual este creado no es suficiente, hay que activar y entrar al entorno virtual, con ls nos va a mostrar que se creó una nueva carpeta llamada venv, y dentro de esa carpeta se encontrara la carpeta Scripts que es donde estará el comando para activar el entorno virtual, para eso colocamos .\venv\Scripts\activate, ahora va a aparecer la palabra venv en la consola:



Para no tener que escribir el comando completo hay que crear un alias, para eso hay que salir del entorno con deactivate, y una vez fuera se coloca alias avenv=.\venv\Scripts\activate.

**Instalación de dependencias con pip:**

Dentro de Python existen muchos módulos, y hay ciertos módulos que no vienen con la instalación por defecto de Python y que hay que instalarlo de manera externa, para manejarlo se utiliza más comúnmente Pip (package instaler for Python) que es un manejador de dependencias, y es un módulo interno que está pensado para instalar módulos que no están por defecto en Python.

Módulos Populares: Requests y BeautifulSoup4 son para WebScraping, Pandas y Numpy se utilizan en la ciencia de datos para trabajar y limpiar datos, Pytests sirve para testing.

Pip normalmente está acompañada del entorno virtual y no debería ser usada fuera del mismo, para utilizarlo hay que entrar al entorno virtual (avenv), aquí una vez se instalen módulos, solo se van a instalar en este proyecto en particular, ahora con pip freeze se ven los módulos instalados. Para instalar pandas se coloca pip install pandas, aquí no solamente se instala pandas, sino que también otras dependencias, esto sucede porque pandas es un módulo muy complejo, y necesita de otras dependencias para funcionar, una vez acabe el proceso ya se puede utilizar libremente, aquí y aparecerá un warning si el módulo esta desactualizado.

Texto

Descripción generada automáticamente

Si se coloca de nuevo pip freeze van a aparecer 5 módulos instalados, pandas y todos los que necesita pandas para funcionar.

¿Y qué pasa si se quiere compartir el proyecto? Pues la persona a la que se le comparte debería tener las mismas dependencias y en las mismas versiones, para lograr eso se usa el comando pip freeze > requirements.txt va a almacenar todas las dependencias con sus respectivas versiones para poder compartirlas, con el comando cat requirements.txt se puede ver que está escrito en el archivo. Cuando llegue la hora de compartirlo, lo único que va a tener que hacer la persona que lo utilice es poner el comando pip install -r requirements.txt y se instalarán exactamente los mismos módulos y en las mismas versiones.

Existen otros instaladores de dependencias como pyenv y pipenv. Estos mejoran un poco las funcionalidades, pero también son más complejos.

**Anaconda:**

Existe una alternativa para instalar dependencias, es un software completo pensado para los científicos de datos. Es una distribución especial de Python de manera gráfica, pero es solo para ciencia de datos, no es la mejor idea hacer backend con esta herramienta.

Para instalarla se va a la página principal de anaconda, ahí se descarga la versión para el sistema operativo que se esté usando, después se ejecutas el instalador, y se le da siguiente, just me, se elige la carpeta, se desmarcan los dos checks que vienen por defecto porque daña el Python que ya está instalado en la computadora, y finalmente install.

Una vez instalado, se ejecuta anaconda.navigator, y se entra a la pestaña Enviroments, le damos create, y elegimos nombre, el lenguaje y la versión del lenguaje. Anaconda ya instala algunas dependencias de base, y para instalar dependencias aparte le damos click en la pestaña para cambiar las dependencias de installed a all, y le damos click al botón Update index, para ver todas las dependencias que se puedan instalas. Finalmente buscamos la dependencia, en este caso pandas, marcamos la casilla y le damos Apply, y esperamos a que el proceso termine.

**Listas y diccionarios anidados:**

Vamos a crear la carpeta oficial del curso, con los mismos pasos que antes y la carpeta va a tener de nombre curso-Python o el que se quiera, y una vez dentro iniciamos Visual Studio Code con code . ahora se inicializa el repositorio de git, y creamos e iniciamos el entorno virtual. Finalmente se ignora el entorno virtual en el repositorio de git.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Si alguna vez se quiere subir el proyecto a GitHub, no se debería llevar todo el entorno virtual, porque dentro de la carpeta venv existe un Python que solo funciona para nosotros.

Las listas son una manera de organizar objetos, y las listas pueden almacenar diccionarios y viceversa. Para hacer esto se crea un nuevo archivo llamado lists\_and\_dicts.py (dentro de VSC se puede hacer directamente), ahí dentro ponemos la función run.

**List Comprehensions:**

Que pasaría su creamos una lista con los primeros 10 números naturales al cuadrado, o de los primeros 100. Para solucionar esto existe un concepto llamado list comprehensions, básicamente es una estructura nueva que contiene corchetes y dentro esta están los elementos de la lista, un ciclo for y un condicional.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Un list comprehension se lee = para cada elemento en el iterable ese elemento solo si se cumple la condición

Datos útiles: El range en un ciclo for tiene su ultimo parámetro de una manera no inclusiva; con ctrl+k+c se comentan todas las líneas seleccionadas y con ctrl+k+u se des comentan

Para cada i en el rango que va de 1 a 101 voy a guardar esa i elevada al cuadrado solamente si i%3 es distinto de 0.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Crear, con un list comprehension, una lista de todos los múltiplos de 4 que a la vez también son múltiplos de 6 y de 9, hasta 5 dígitos: [36, 72, 108, 144, 180, 216, …]

**Dictionary Comprehensions:**

Existe la contraparte de las list comprehensions en los diccionarios, y la estructura es la siguiente:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Para cada elemento en un iterable yo voy a colocar una llave y un valor solamente si se corresponde a una condición.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Para cada i en el rango de 1 al 101 voy a guardar a i como llave y a i elevado al cubo como valor solo si i%3 es distinto de 0.

Crear, con un diccionary comprehension, un diccionario cuyas llaves sean los primeros 1000 números naturales con sus raíces cuadradas como valores.

**Funciones anónimas: lambda**

Las funciones son una manera de reutilizar código que después puede ser usado en diferentes secciones del proyecto, así solo se escribe una vez el código, para hacerla solo se pone def (nombre): .Existe una manera distinta de crear funciones sin nombre, o más bien funciones anónimas conocidas en Python como lamda, y tienen una serie de características particulares, su estructura es:

Imagen que contiene Texto

Descripción generada automáticamente

Las funciones lambda pueden tener los argumentos que se necesiten, pero solo va a ser una línea de código, en otros lenguajes como java script si pueden tener más líneas de código, en estas funciones no hace falta colocar la palabra return. Un buen ejemplo de una función lambda en el código para verificar si un string es palíndromo:

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente con confianza media

En el ejemplo se observa un argumento, que es el parámetro que recibe la función para trabajar, luego la expresión que es una línea de código y una variable que tiene un identificador, pero no es de la función, sino de una variable que contiene un objeto tipo función que retorna toda la expresión. Como lambda es una función sin nombre, la forma de llamarla es con el nombre de la variable que va a guardar el objeto de tipo función que esta expresión retorna.

Aquí uno no llama una función en si misma, sino que se utiliza la variable con los paréntesis para invocar al objeto de tipo función que esta almacenado en la variable. Así se vería el mismo código, pero con una función normal:

Texto

Descripción generada automáticamente con confianza media

**High order functions: filter, map y reduce:**

Una función de orden superior es aquella que recibe como parámetro otra función, y tiene una seria de características especiales, por ejemplo:

Texto

Descripción generada automáticamente con confianza media

Cuando nosotros llamamos a saludo, saludo recibe la función que le pasamos como parámetro y la ejecuta dentro de si misma, por lo tanto, si ejecutamos saludo pasándole como parámetro la función hola, el resultado va a ser hola, y si se lo pasamos con la función adiós, el resultado va a ser adiós. Existen tres tipos de funciones de orden superior, la primera se llama filter, en el siguiente ejemplo se busca obtener solo los números impares de la lista, por lo tanto, tengo una lambda función que recibe cada uno de los elementos de la lista de ese iterable que es el segundo parámetro de la función filter, y ese resultado lo convierto en lista con la función list.

Texto

Descripción generada automáticamente

La siguiente función de orden superior es map, en este ejemplo se busca es convertir una lista a otra con los números elevados al cuadrado, aquí tengo mi función lamda como mi primer parámetro, tengo mi lista como segundo parámetro, y después envuelvo todo esto en la función list que esta almacenada en la variable squares.

Texto

Descripción generada automáticamente

La última función se llama reduce, y para poder usarla hay que importarla del módulo functools porque no viene por defecto en Python, en este ejemplo se busca multiplicar todos los elementos de una lista o sea reducir los valores de la lista a un único valor. Este ejemplo funciona tengo mi lista de datos llena de números dos, y lo que hago es guardar dentro de la variable all\_multiplied el resultado de hacer reduce de la función lambda como primer parámetro y my\_list como segunde parámetro, ahora lambda lleva dos parámetros y los multiplica, estos a y b en la primera iteración son el primer y segundo elemento de nuestra lista o sea 2\*2=4, en la segunda iteración ese 4 va a pasar a ser el parámetro a y b va a ser el tercer elemento de la lista, este ciclo se repite hasta que se acaben los elementos.

Texto

Descripción generada automáticamente

**Filtrando datos:**

A partir de una lista de diccionarios hay que filtrar los datos, o sea hay que escribir diferentes funciones de orden superior para filtrar los datos. Cuando una variable esta en mayúsculas, quiere decir que no se espera modificarla, o sea una constante. La lista de diccionario tiene la misma estructura, una llave nombre, la edad, la organización, la posición y el lenguaje, y a partir de eso se crean diferentes filtros.

Esta línea de código funciona así: Tengo un list que lo que hace es convertirme en lista el resultado de la función map, la función map tiene dos parámetros, una función lambda y un iterable, lo que se está haciendo es transformar a cada uno de los diccionarios que tenemos en data en un diccionario nuevo que es la combinación de nuestro diccionario original, con un nuevo diccionario, que simplemente es una llave old y un valor que es el resultado de la expresión. Es decir, al evaluar que la edad es mayor a 70 obtengo true, el resultado va a ser true y la llave old va a contener ese valor, pero si al evaluar que es menor a 70, el resultado va a ser false y la llave old va a contener ese valor. El símbolo se denomina pipe y lo que hace es unir a un diccionario con otro nuevo.



Proyecto: crear las listas all\_python\_devs y all\_Platzi\_workers usando una combinación de filter con map. Y crear old\_people y adults con list comprehensions.

**Errores en el código:**

Existen dos casos de errores, el primero es cuando Python avisa que existe un error y cuando no. Cuando Python avisa que existe un error, devuelve un tracepack, y se dividen de la siguiente manera:

Diagrama

Descripción generada automáticamente

Los syntaxerror detiene el programa, porque existe un typo (error de escritura), los otros errores son excepciones, estas si hacen que quiebren toda la lógica, y cuando Python detecta una excepción, lo hará en una línea específica y las demás líneas de código si se ejecutaran.

* KeyboardInterrupt y sucede cuando pulsamos ctrl+c en la consola interactiva de Python y todo se cierra, Python lo que hace es elevar, lo que quiere decir que crea un objeto de tipo excepción, y lo va moviendo en el programa desde dentro hacia fuera.
* El KeyError sucede cuando intentamos acceder a una llave de un diccionario que no existe.
* IndexError es cuando se está trabajando con listas y se intenta acceder a un índice de esa lista que no existe.
* FileNotFoundError sucede cuando se intenta abrir un archivo que no existe.
* ZeroDivisionError sucede cuando se intenta dividir un numero entre 0.
* ImportError sucede cuando intentamos importar y hay algún error en el módulo y falla.

Cuando Python muestra una excepción lo hace con un mensaje denominado traceback, y se lee de forma correcta desde el final hasta el principio, porque la última línea muestra la excepción que sucedió y que significa, en la penúltima línea muestra el archivo en el que sucedió el error, y el segundo elemento muestra en que línea sucedió, y el último elemento es el módulo. Finalmente, la antepenúltima línea muestra traceback, y explica de donde parte el error, (dice la llamada mas reciente esta al final) porque cuando la excepción ocurre lo hace dentro de un lugar específico, si el error no fue capturado, lo que va a hacer Python es elevarlo a una función superior, por ejemplo su se trabaja en run y dentro se ejecuta otra función, si dentro de esa función sucede la excepción, Python lo que hará es llevar el error a la función run y si no se trabaja el error ahí, se corta la ejecución del programa y muestra un traceback.

**Debugging:**

También llamada depuración, es una herramienta que tienen los editores de código para resolver los errores, para usarla hay que darle en run and debbug en visual studio code. Al ejecuitarle va a aparecer un nuevo menú con opciones, pausa, paso siguiente, meternos dentro, salir de un luger, reestart, detener. Visual studio code va a mostrar la línea de código que se está recorriendo y en la cual se le pone pausa, de esta forma se puede recorrer el programa línea por línea para encontrar el error.

Un breakpont es un punto de quiebre, que sirve para pausar en un punto especifico, y se representara con un punto reojo a la izquierda. Esta es una herramienta muy importante para hacer un proceso de depuración.

**Manejo de excepciones:**

Try y except significa “probar” y “excepto que hago lo siguiente”, supongamos que tenemos un programa que comprueba si un string es palíndromo, en donde el trace back nos dice en donde sucedió el error y la expresión que está dando el error. Ahora hay que evitar que el usuario ingrese un número, de manera él try con el print ejecuta la función palindrom y si eso falla, Python corta el programa ejecuta la línea except con el mensaje:

Texto

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto

Descripción generada automáticamente

Siguiendo con el mismo ejemplo, supongamos que el usuario ingresa una cadena vacía, el resultado va a ser true, pero no se le debería permitir al usuario hacer eso, para eso se utiliza raise, ahora arriba en la definición de la función tenemos un try except que envuelve a todo el código de la función, en el try se pregunta si la longitud del string es 0, si es así, entonces se eleva una excepción de tipo value error y pasar el mensaje de error, en caso contrario se retorna el resultado de la expresión, por otra parte en el except sucede que excepto que suceda un value error al que se le nombra ve, se ejecuta un print con ese error, el mensaje va a ser el raise y retorna un false.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

La última palabra es finally, y se usa al final de una escritura try except para cerrar un archivo, cerrar una conexión a una base de datos o liberar recursos externos. Su función es hacer algo si sucede un error o no.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Desafio: utiliza las palabras clave try, except y raise para elevar un error si el usuario ingresa un numero negativo en el programa de divisores.

**Assert Statement:**

En Python existe una última forma de manejar errores, son expresiones combinando variables con operadores como en cualquier expresión, el flujo es el siguiente:

Diagrama

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

En la estructura se utiliza la palabra clave assert, luego una condición que es la que devuelve verdadero o falso, y finalmente un mensaje de error. Un buen ejemplo se puede ver de nuevo en el programa palíndromo:

Texto

Descripción generada automáticamente

En este programa al ingresar una cadena String vacia va a devolver un assention error que es un tipo de error especial que tienen los asserts para poder manejar el flujo del programa, este error viene acompañado del mensaje de error que se le coloque.

El método isnumeric() es un método que lo que hace es devolver verdadero si esa string corresponde a un número y falso si no corresponde. Es un método de las cadenas de caracteres.

Desafio: Utiliza assert statements para evitar que el usuario ingrese un numero negativo en nuestro programa de divisores.

**Manejo de archivos:**

Existen diferentes tipos de archivos, y se clasifican en archivos de textos que tienen bytes que solo representan letras y símbolos, en cambio los archivos binarios tienen bytes que representan cosas mucho más complejas. En Python no se trabaja mucho con archivos binarios, estos se trabajan con programa directamente dedicados a estos archivos, pero con Python por lo general solo se utilizan archivos de texto.

Gráfico, Gráfico circular

Descripción generada automáticamente

El json es de desarrollo web, que es una manera de compartir información entre diferentes servicios cuando se desarrolla una aplicación web. El py es el que normalmente se utiliza en python. El xml también se utiliza para compartir información, el css y js son archivos para crear paginas web. Y el csv se utiliza en ciencia de datos.

Existen tres modos de apertura en Python, lectura R, escritura W (sobrescribir), y el modo escritura A (agrega al final). La palabra clave with es un manejador contextual, lo que hace es un control del flujo del archivo haciendo que, si se cierra el programa, el archivo no se rompe. La función open abre el archivo y siempre va a llevar la ruta del archivo en formato String y un segundo parámetro que es el modo de apertura, finalmente as nos sirve para darle un nombre hipotético al archivo dentro del programa, por lo que dentro del programa se refiere al archivo con ese nombre. Un ultimo parámetro que es encoding=”utf-8” es para asegurarse de que todo lo que se vaya a escribir no tenga caracteres extraños.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

**Reto final:**

Hay que hacer le juego del ahorcado, el archivo se va a llamar juego\_del\_ahorcado.py, aquí limpia la pantalla y va a preguntar adivina la palabra por letra. Con manejo de errores y manejos de archivos con data.txt que contendrá la base de datos del juego. Investigar la función enumerate, el método get. La línea os.system(“cls”) en windows y os.system(“clear”) en unix es para limpiar la pantalla.

Se puede poner un sistema de puntos, y dibujar el muñeco con el código ascii.