# Python

Clase 23/1

Plataformas que se van a usar en el curso:

Playground  plataforma donde se encuentra el material y donde se cargan las clases grabadas

Discord  consultas fuera del horario de clases

Zoom  clases

Se presenta el curso y aclara los ejercicios introductorios.

Recomienda practicar programación una vez finalizado el curso.

Contenido teórico + Práctica en casa  dedicación de 15hs semanales

Nivelación:

Programación en Python  ver el curso de Python 3

Programación tiene 2 caras: pensar en cómo resolver el problema + decirle al programa cómo hacerlo

Leer para la próxima clase el material asincrónico.

Instalar Python 3.8/3.9 + Material asincrónico

Clase 26/1

En Anaconda Prompt 🡪 jupyter lab y se abre el notebook en el browser

En las notebooks de JupyterLab el print es automático.

Strings

[] sirve para indicar la posición que se desea imprimir del carácter, ejemplo: “Digital”[0] 🡪 D

[-1] cuenta desde atrás

Para concatenar strings 🡪 +

“Hola” +” ”+ “Celina” 🡪 Hola Celina

Para contar cuantos caracteres tiene una cadena se usa len()

Buscar los errores en StackOverFlow, w3school son plataformas recomendadas.

Se crean variable con = 🡪 ejemplo: a = 8

Los nombres de variables van a la izquierda, mayúsculas son distintas de minúsculas, siempre deben empezar con letras sin caracteres especiales.

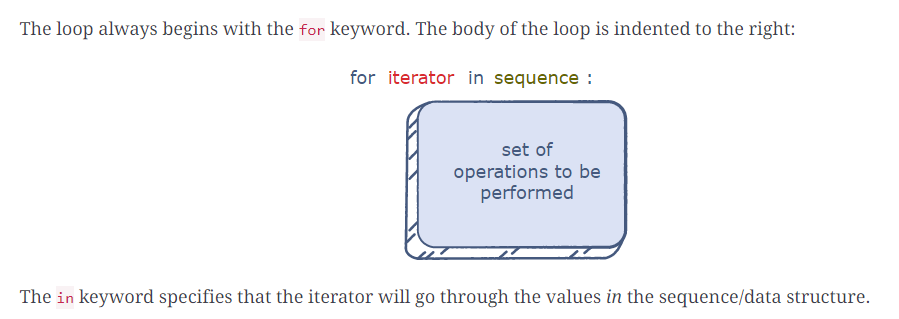
Existen palabras reservadas que no se pueden usar en nombres de variables.

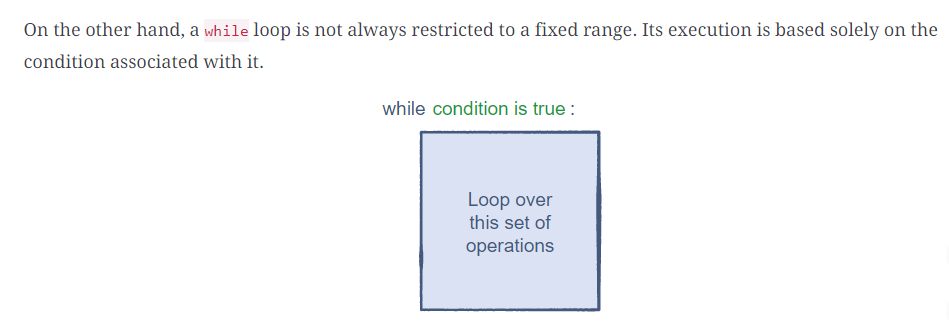
Próxima clase: resolver la notebook C2 en Jupyter y ver el contenido en Variables y Funciones en Playground.

Para abrir un script en Anaconda prompt tengo que cambiar el directorio con el comando cd 🡪 cd C:\Users\Celina\Digital\_House

Todos los condicionales deben finalizar con un símbolo de dos puntos (:), para indicar que estamos ingresando a un bloque de código especifico. 🡪 if, elif, else

La **tupla** es una estructura de datos parecida a las listas. Para definir una tupla es necesario declararla entre paréntesis. Al igual que en las listas, los elementos dentro de una tupla se separan con comas, y podemos acceder a ellos a partir de su índice o haciendo slicing. En las tuplas no es posible realizar un cambio en sus elementos sin que internamente sean destruídas y vueltas a construir. Por lo cual, generalmente se suelen agrupar datos estáticos en estas estructuras, que sabemos que no vamos a necesitar modificar. Por otra parte, tampoco es posible realizar una operación de asignación.





Recursion is useful when we need to divide data into different chunks. Iteration is useful for traversing data and also when we don’t want the program’s scope to change.

La estructura de control if... elif...else... nos permite encadenar varias condiciones. elif es una contracción de "else if". La orden en Python se escribe así:

if condicion\_1:

bloque 1

elif condicion\_2:

bloque 2

else:

bloque 3

Para tener en cuenta:

🡪Si se cumple la condición 1, se ejecuta el bloque 1.

🡪Si no se cumple la condición 1 pero sí que se cumple la condición 2, se ejecuta el bloque 2.

🡪Si no se cumplen ni la condición 1 ni la condición 2, se ejecuta el bloque 3.

Los **operadores lógicos** permiten unir un conjunto de evaluaciones distintas en una sola expresión más compleja. En Python, estos operadores son:

not: Niega una expresión.

and: Evalúa potencialmente dos expresiones; si ambas son verdaderas, devuelve True, pero si alguna o ambas son falsas, devuelve False.

or: Evalúa potencialmente dos expresiones; si alguna es verdadera, devuelve True, pero si ambas son falsas, devuelve False.

En Python, lo que determina la forma en la que se encuentra estructurado el código es la **indentación**. Podemos pensar en la indentación como la sangría que debemos dejar al inicio de ciertas líneas de código. Por lo general, esta indentación contiene cuatro espacios ("\s\s\s\s") o una tabulación ("\t").

Todos los bloques están delimitados por la profundidad de la sangría y, por lo tanto, su importancia es vital para un programa de Python.

Los **bloques** son una o más instrucciones que deben ejecutarse una tras otra, de arriba hacia abajo abajo, de izquierda a derecha. Existen varios tipos de bloques, los más comunes son bloques de código, es decir, bloques que contienen instrucciones de Python. Otro tipo común de bloque son los bloques de comentarios, es decir, un conjunto de caracteres, que ocupan una o más líneas de código y están delimitados por una notación que ha definido el lenguaje de programación. En Python, podemos "comentar" el texto utilizando el símbolo numeral (#).

El símbolo **%** en Python se llama el Operador de Módulo. Retorna el remanente de la división del operando izquierdo por el operando derecho. Se usa para obtener el residuo de un problema de división.

+: Suma

-: Resta

\*: Multiplicación o producto

/: División

//: Floor division

%: Resto de la división o módulo

\*\*: Potencia

Ejemplo de uso de la función def:

**def** duplicar(numero):

n **=** numero

doble **=** n **\*** 2

**return** doble

duplicar(3)

También se puede hacer un print del resultado de la función, ejemplo: resultado = print(duplicar(4)).

Print(duplicar(4)==3) 🡪 False

Un recurso muy importante en Python son las **listas por comprensión**. Esta forma de estructura de datos permite realizar iteraciones, operaciones sobre datos, operaciones lógicas y, al mismo tiempo, definir una lista que contenga los resultados esperados.

La utilidad de las listas por comprensión viene dada por la simpleza de su sintaxis: ¡basta una sola línea de código para definirlas! A su vez, las listas por comprensión suelen ser muy eficientes en términos de recursos computacionales.

Ejemplo: # Definimos una lista

enteros **=** [1, 2, 3, 4, 5]

# Definimos una lista por comprensión

enteros\_mas\_uno **=** [numero **+** 1 **for** numero **in** enteros]

**print**(enteros\_mas\_uno)

De forma genérica, la sintaxis de las listas por comprensión es:

[\*\*expresión\*\* for \*\*elemento\*\* in \*\*iterable\*\*(if\*\*condición\*\*)]

La condición if es opcional.

Veamos la sintaxis general para la definición de un **diccionario por comprensión**:

{\*\*expresión\_para\_clave\*\* : \*\*expresión\_para\_valor\*\* for \*\*elemento\*\* in \*\*iterable\*\* [if \*\*condición\*\*]}

La condición if es opcional.

Vamos a explicar de forma ordenada cómo funciona la sintaxis de los diccionarios por comprensión:

expresión\_para\_clave: es una expresión que define a la clave; debe dar como resultado una clave diferente durante cada iteración, ya que en un diccionario nos encontramos con la restricción de que las claves no deben repetirse.

: : los dos puntos separan la expresión de la clave de la expresión que da lugar al valor asociado.

expresión\_para\_valor: es la expresión que define el valor para una clave dada; funciona igual que en las listas por comprensión.

for: da inicio a un bucle, al igual que en las listas por comprensión.

elemento: es el nombre de la variable a la que asignamos cada elemento del iterable durante cada iteración (igual que en las listas por comprensión).

in: indica sobre qué vamos a iterar.

iterable: es el objeto que vamos a recorrer.

if: señala la expresión condicional que vamos a evaluar sobre los elementos del iterable que recorremos (al igual que en las listas por comprensión, su inclusión resulta opcional).

condición: es el condicional que va a ser evaluado.

Ejemplo: animales = ['ballena', 'elefante', 'gallina']

mi\_diccionario = {x:len(x) for x in animales}

También existen lo que llamamos **"Truthy" y "Falsy"**. Traducido al pie de la letra sería "cercano a Verdadero" y "cercano a Falso". Estos son los otros valores (o tipos de datos) que podemos usar como verdadero y falso. Para eso usaremos la función bool(), que nos va a ayudar a convertir algún tipo de dato a un booleano.

Clase 30/1

Para extraer caracteres de un str: S[], donde S es la variable del str; también se puede extraer utilizando : 🡪 S[1:]

Similar a una función es un método:

objeto.método(argumentos)

A veces se necesitan descargar librerías, para eso se llama con import

import math

Entonces se podrá usar math.[función]

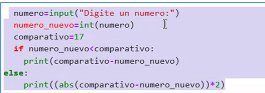
O 🡪 from math import \* 🡪 trae todas las funciones de la librería, y ya no es necesario escribir math.función, directamente se escribe la función.

Input() siempre devuelve str

Próxima clase: resolver notebook C3, + ejercicio que manda Paula.

Una función siempre empieza con DEF y termina con RETURN, sin print() intermedio.

Clase 2/2

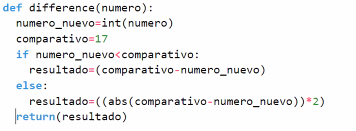


Def nombre\_funcion (argumento):

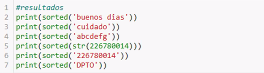
Operación

Return lo que quiero que me devuelva la función

Resolución Ejercicio 9, Desafío Data Science:



Ejercicio 11 del C3



Acceder a los elementos de una lista

Se puede acceder a cualquier elemento de una lista usando index(), o print(lista[posición]).

Unir lista con append().

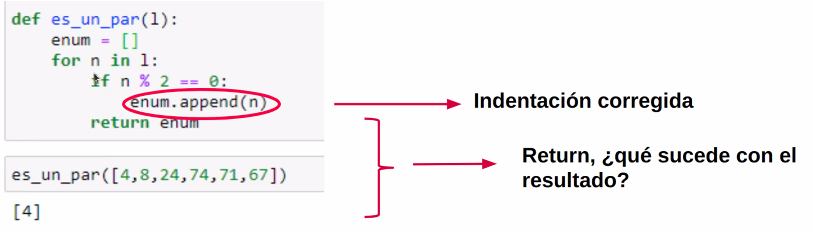
Insertar un elemento en una posición de una lista, insert().

Range() es una función que devuelve una lista de enteros que se pide de acuerdo a un paso.

Even\_num = range(1,11,2) 🡪 da los números pares del 1 al 10 (no se incluye el 11)

If + condición … + return (obligatorio) para cerrar el comando

Clase 6/2



En el ejemplo, le falta indentación al return para que devuelva la lista completa de pares. Como la palabra clave esta dentro del for, me devuelve el primer valor de la lista.

Para que el resultado nuevo no se pise cada vez que corre la función, hay que crear la variable fuera del comando for y dentro de la función.

Existe una función LAMDA, revisar el ppt y buscar biblio.

Es diferente pedir las referencias por VALOR o por COPIA.

Revisar operaciones con diccionarios, hay ejemplos en el ppt y en el notebook C5.

Links de interés

<https://docs.python.org/es/3/library/functions.html>

<https://www.hackerrank.com/domains/python>

<https://exercism.org/tracks/python>

<https://edabit.com/challenges/python3>

<https://www.codewars.com/kata/python>