Curso de Python para Analisis de Datos

Clase 1

Indice:

Introducción a la programación:

Que es la programacion?

Entendemos por programación la actividad de **programar** ; y por programar , el decirle a una computadora —o, mejor dicho, cualquier dispositivo programable— lo que tiene que hacer. De modo que la programación es un tipo de **comunicación**, comunicación entre un hombre (el programador) y una máquina (la que es programada). Y como todo tipo de comunicación, precisa de un **lenguaje** que sea comprendido tanto por el emisor como por el receptor. A ese lenguaje lo denominamos lenguaje de programación . El **mensaje** codificado según las reglas de un lenguaje de programación lo llamamos programa o aplicación.

Existen muchos **lenguajes de programación**, esto es, como dijimos, medios de comunicación entre un programador y una computadora. Cada uno de ellos se caracteriza por algo en particular: en el proceso de **"instruir"** a una máquina para que ejecute las acciones que le indicamos, algunos lenguajes son más óptimos o acordes que otros para tal o cual tarea. Así, ciertos lenguajes están pensados para "hablarle" a un celular; otros, a una computadora de escritorio; otros, a un microondas. Y algunos son más fáciles de aprender, del mismo modo como decimos que el griego y el árabe son menos accesibles que el inglés. El inglés, no obstante, es más acorde si nos queremos dedicar al comercio internacional; así como el griego lo es para el estudio de los pensadores clásicos de la antigüedad.

Pero seamos un poco más precisos: no es cierto que las computadoras comprendan las instrucciones que les comunicamos vía un lenguaje de programación. De hecho, solo están diseñadas para entender un dialecto al que se denomina **código máquina** . Pero escribir un conjunto de instrucciones directamente en este código es simplemente imposible para un hombre. Por ende, necesitaremos un **"traductor"** (que no es sino un programa) que convierta nuestras instrucciones escritas según las reglas de un lenguaje de programación en particular a código máquina.

Python es un lenguaje de programación. Pero también llamamos Python al programa que traduce Python-el-lenguaje a código máquina para que pueda ser interpretado por una computadora. De ahí que en la expresión "instalar Python" nos referimos a Python-el-programa, mientras que "programar en Python" se refiere a Python-el-lenguaje. En la jerga, para evitar dicha ambigüedad, a Python-el-programa también se lo denomina intérprete .

Python es un lenguaje que se caracteriza por tener una **sintaxis sencilla** (fácil de aprender para legos en programación) y **clara** (elocuente y de lectura fluida). Por ello resulta una herramienta ideal para iniciarse en esto de la programación. Pero independientemente del lenguaje que usaremos en este curso, lo más importante es que aprenderemos conceptos generales de la programación (aunque claro, con las particularidades de Python), que se aplicarán igualmente para otros. Así como aprender un segundo idioma es más sencillo habiendo dominado nuestra lengua materna, del mismo modo ocurre en el mundo de la programación. Y no es de extrañar que una misma persona, con el tiempo, acabe versada en varios lenguajes de programación; aunque por lo general uno termina tomando una decisión según sus propios gustos y especializándose en alguno en particular.

Para finalizar este apartado, precisemos una clasificación que nos será útil para poder avanzar. Ya dijimos que un programa o aplicación es un conjunto de instrucciones escritas según las reglas de un lenguaje de programación. Si bien existen varios tipos de aplicaciones, en este curso estaremos trabajando con los siguientes dos:

- Aplicaciones de consola
- Aplicaciones de escritorio

Por el momento basta con saber que en Python los programas son, por defecto, aplicaciones de consola. Estructura de un programa Me permito usar la siguiente analogía: escribir un programa es como redactar una receta de cocina. La diferencia es que quien escribe es un programador y quien “cocina”, una máquina. Las computadoras, a diferencia de los cocineros, nunca se equivocan al interpretar una “receta”. Siempre hacen lo que le decimos, al pie de la letra. Todos los problemas que surgen en la industria del software radican en creer que le estamos indicando una actividad X a una máquina cuando, en realidad, programamos una actividad Y. Esto es, cuando hay una disociación entre lo que le “decimos” y lo que quisimos decir.

De esta primera definición se sigue que podemos escribir un programa tanto en un pizarrón como en un programa pensado para esa tarea (al que llamamos editor de código). Las instrucciones de los programas tienen una disposición similar a la del español: se interpretan de arriba a abajo y de izquierda a derecha. Aunque a veces podemos indicarle que queremos que tal instrucción o conjunto de instrucciones se ejecute dos o más veces, o que en ocasiones no se ejecute, o que se vuelva a la primera instrucción, entre otras cosas similares. Este recorrido del código (entendemos código por conjuntos de instrucciones) que hace una máquina al interpretarlo se denomina flujo de un programa .

Un conjunto de instrucciones con el objetivo de obtener un determinado comportamiento se conoce como algoritmo (según el diccionario de la Real Academia Española: conjunto ordenado y finito de operaciones que permite hallar la solución de un problema). Por ejemplo, el proceso por el cual se calculan las raíces de una ecuación de segundo grado constituye un algoritmo.

Por que Python?

Uno de los motivos por el cual Python ha crecido en los ultimos años es su gran comunidad de código abierto. Existen muchos eventos a los que se puede asistir para aprender más acerca de lo que se trabaja en Python. Todos ellos son inclusivos y abiertos al público.

Algunos ejemplos:

Conferencias
PyCon Argentina (https://twitter.com/pyconar/status/1147119669049999364). Conferencia Anual de Python organizada por PyAr (https://www.python.org.ar/), comunidad Python Argentina.
PyData Argentina (https://pydata.org/cordoba2019/). Evento enfocado a Analisis de datos y Data science utilizando Python, R y Julia.

MeetUps
Data Science Argentina (https://www.meetup.com/es/Argentina-Data-Science-Meetup/)
PyData Buenos Aires (https://www.meetup.com/es/PyData-Buenos-Aires/)
Women in Machine Learning Buenos Aires (https://www.meetup.com/es/Buenos-Aires-Women-in-Machine-Learning-Data-Science/)

Python es un lenguaje de programación de propósito general muy poderoso y flexible, a la vez que sencillo y fácil de aprender.

¿Donde recurro cuando estoy trabada o no se qué hacer?

Python posee una amplia documentación que nos permite investigar cada función que contiene el lenguaje. Podes chequearla acá (https://docs.python.org/3/)

Otra fuente para obtener respuestas es Stackoverflow (https://es.stackoverflow.com/) o por supuesto, Google.

Python y Data Science

Utilizar Python para la Ciencia de Datos se transformó en una de las elecciones mas populares en el mundo para comenzar a andar esta nueva carrera profesional. La eleccion del lenguaje de programacion adecuado para proyectos de grandes cantidades de datos es una desicion crucial, ya que migrar un proyecto una vez que comienza es muy dificil. Aunque hay varios lenguajes que tambien se utilizan con Datos, como R, Java o Sas, la preferencia hacia Python se basa en 5 grandes razones para elegirlo.

Simplicidad: Python es conocido por hacer que los programas funcionen en la menor cantidad de lineas de codigo. Identifica y asocia automaticamente los tipos de datos y, en general, resulta un lenguaje facil de usar y toma menos tiempo en la codificacion. Tampoco hay limitacion para el procesamiento de datos. Puede calcular datos en cualquier tipo de equipo y entorno, básicamente en todas partes. Anteriormente se argumentaba que Python era más lento que algunos de sus homólogos como Java y Scala, pero con la plataforma Anaconda se ha puesto al día demostrando que es rápido tanto en desarrollo como en ejecución.

Compatibilidad: Hadoop es la plataforma de big data de código abierto más popular y la compatibilidad inherente de Python es otra razón más para preferirlo a otros lenguajes. Cuando hablamos de Data Science siempre vamos a estar relacionados con Big Data, y por ello tambien es importante la afinidad de ambas plataformas.

Facilidad de aprendizaje:En comparación con otros idiomas, Python es fácil de aprender incluso para los programadores con menos experiencia. Es un primer idioma ideal debido a tres razones principales: cuenta con amplios recursos de aprendizaje, garantiza un código legible y se rodea de una gran comunidad. Todo esto

se traduce en una curva de aprendizaje gradual con la aplicación directa de conceptos en programas del mundo real. La gran comunidad Python ofrece la seguridad de saber que, caso de encontrar problemas en el desarrollo, habrá otros que puedan echar una mano para ayudar a resolverlos.

Paquetes de gran alcance:Python tiene un poderoso conjunto de paquetes para una amplia gama de necesidades de análisis y ciencia de datos. Algunos de los paquetes populares que le dan a este idioma una ventaja son NumPy, Pandas, Scipy, Scikit-learn, PyBrain, Tensorflow, Cython, PyMySQL, BeautifulSoup o iPython.

Visualizacion de datos:Aunque R es mejor en lo que respecta a la visualización de datos, con paquetes recientes, Python para big data ha mejorado su oferta en este espacio. Ahora existen API que pueden ofrecer buenos resultados.

Python es un lenguaje muy popular como puede comprobarse en cualquier equipo de científicos de datos. Siempre es fácil encontrar algunas personas en cada departamento como marketing, desarrollo, mantenimiento, servicio al cliente con un conocimiento práctico de Python, lo que supone el mejor seguro para las empresas. No siempre es fácil establecer una comunicación entre los diferentes departamentos y, con Python y big data, este tipo de inconvenientes no existen.

Empecemos!

Instalación de Python y Jupyter Notebook

Antes de empezar a programar debemos instalar las aplicaciones necesarias para esta tarea. Lo principal para este curso es tener instalado Anaconda (lo cual nos va a instalar todo lo que necesitemos para comenzar nuestra carrera de Data Scientist). Para ello vamos a dirigirnos a la página oficial del programa (https://www.anaconda.com/distribution/ (https://www.anaconda.com/distribution/)) y seguir los pasos.

Importante:

A la hora de instalarlo no deben olvidar seleccionar la version **Python 3.X**, ya que las versiones anteriores ya no tienen soporte.

Aclaraciones

Anaconda es un entorno de trabajo que nos instala varias cosas al mismo tiempo: La ultima version de Python, el Jupyter Notebook (Nuestro IDE), las librerias que vamos a necesitar durante nuestros comienzos, entre otras herramientas que ya veremos a lo largo de este curso.

En estas clases utilizaremos **Jupyter Notebook**. La ventaja de jupyter notebook es que permite mezclar celdas `Markdown` con celdas de código.

1) **Markdown** es un lenguaje de marcado que permite dar formato a un texto de manera rápida y fácil.

Para utilizarlo debes seguir ciertas convenciones que puedas consultar en la siguiente página web: Guía Markdown (https://joedicastro.com/pages/markdown.html)

2) Por otro lado, en las celdas de código es donde vamos a escribir el código **Python**.

Prestale atención a los corchetes que se encuentran al lado de cada celda de Python. Cuando tienen un número significa que esa celda ha sido corrida. Cuando tienen * implica que la celda esta corriendo.

Python posee una guía de estilo, que si bien no es obligatoria, se **RECOMIENDA** seguir. Podes leer acerca de las convenciones establecidas en el PEP8 (https://www.python.org/dev/peps/pep-0008/).

El simbolo # se utiliza para determinar un comentario. Esto significa que al verlo, Python entendera que esa línea no corresponde a código que debe correrse.

Comentar el código es una practica muy útil para explicar que hace una linea de código. La convención es que el comentario se haga en la línea anterior a la linea de código correspondiente.

La filosofía de Python hace hincapié en una sintaxis que favorezca un código legible. Esta filosofía se resumen en lo que se conoce como el **ZEN de Python**.

In [ ]:
```
#Corre esta linea para descubrir el zen de Python
import this
```

Empecemos con una función muy simple de Python: `print()`. Esta función imprime la variable que se le pase.

In [ ]:
```
#Corre esta celda para descrubir el output
print("Python es muy fácil de aprender")
```

Prueba modificar el texto dentro de las comillas para poder editar lo que quieres imprimir

Expresiones

Avancemos un poco más con el diseño del código. Todos los lenguajes de programación tienen la capacidad de ejecutar `operaciones matemáticas simples` (suma, resta, multiplicación, división); es decir, incorporan una suerte de calculadora virtual que podemos usar en nuestros programas para hacer este tipo de operaciones. La `sintaxis` (por sintaxis entendemos el orden y la relación de los términos de un lenguaje necesarios para que una instrucción sea comprendida) para realizar una suma en Python es similar a la que usamos en matemática para expresar dicha operación.

In [ ]:
```
#Ejecuta esta linea!
print(5 + 7)
```

Python puede usarse para hacer calculos básicos. Veamos alguna de las funciones que pueden ser utilizadas en Python

In [ ]:
```
#Suma
print(4+3)
#Resta
print(5-2)
#Multiplicacion
print(3 * 5)
#División
print(10 / 2)
```

In [ ]:
```
#Corre las siguientes celdas y descubre que funciones se utilizan
print(4 ** 2)
print(18 % 7)
print(7//2)
```

EXPLICACION

La primera funcion es la potenciación esto es 4^2

El operador % devuelve el resto de la division, en este caso entre 18 y 7 (llamado `modulo`)

El operador // devuelve el cociente de la division entre 7 y 3

Variables y Estructura de datos

Asignación de variables:

En Python, una variable permite referir a un valor utilizando un nombre en especifico. Para crear una variable, se debe usar = como en el siguiente ejemplo:

In [ ]:
```
b = "Hola mundo"
```

Ahora, sigue las instrucciones de los comentarios.

In [ ]:
```
# Ejecuta para invocar a la variable
b
```

In [ ]:
```
#Asigna 100 a la variable mi_variable
mi_variable = 100
```

In [ ]:
```
#Muestra el contenido de la variable
print(mi_variable)
```

In [ ]:
```
resultado = factor**7* mi_variable
print("Hola "+ "Mundo")
```

Las variables pueden ser llamadas posteriormente. Además podemos realizar calculos y operaciones con ellas.

In [ ]:
```
#Crea una variable llamada factor que contenga el numero 1.1
factor = 1.1

#Calcula el resultado de elevar a la 7 la varible factor multiplicarla
#por el numero contenido en mi_varible y asignala a una nueva variable
#llamada resultado
resultado = factor**7*mi_variable
#Imprime el resultado
print(resultado)
```

Hay ciertas reglas que se deben cumplir a la hora de declarar una variable:

- Los nombres de las variables siempre aparecen a la izquierda de =.
- Python diferencia mayusculas de minusculas en el nombre de las variables.
- Los nombres de las variables DEBEN comenzar con una letra. Luego pueden contener nombres y guiones bajos _. Pero no pueden contener caracteres especiales (como por ejemplo, &, *, #, etc).
- Si bien Python no le importa como llames a tus variables, los nombres de las variables deben ser descriptivos de los valores o datos que contienen para que otras personas puedan interpretarlo.
- Lee más acerca de estas convenciones en el **PEP8**.

PALABRAS RESERVADAS: Hay 33 palabras que no podes utilizar como nombres de variables, debido a que estan reservadas para el lenguaje ya que cumplen una función particular en Python.

Ellas son: False, None, True, and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

Entrada de datos

Hasta el momento hemos estado usando la instrucción print para imprimir información en la consola. La operación inversa es la entrada de datos, esto es, que el usuario ingrese información en la consola y podamos guardarla en una variable.

En Python esto se realiza vía la instrucción input .

In [ ]:
```
#Ejecuta esta linea
nombre = float(input("Escribe un numero: "))
resultado = nombre + 2
resultado
```

Notarás que input es similar a print , puesto que lo que indicamos entre paréntesis y entre comillas será mostrado en pantalla. Pero a diferencia de ésta, de input decimos que “retorna” un valor (así como 7 + 5 retornaba 12), que no es otra cosa que lo escrito en la consola por el usuario, y lo almacenamos en una variable que queramos.

Una vez ingresado el nombre podemos imprimirlo en pantalla:

In [ ]:
```
#Ejecuta esta linea
nombre = input("Escribe tu nombre: ")
print(nombre)
```

También, como el resultado de input es siempre una cadena, podemos usar el operador de suma para mostrar un saludo:

In [ ]:
```
#Ejecuta esta linea
nombre = input("Escribe tu nombre:")
print("Hola " + nombre)
```

Tipos de dato

Los tipos de dato permiten clasificar la información que contiene una variable. En Python, distinguimos cuatro tipos de dato básicos: números enteros(`int`), números de coma flotante (`float`), cadenas de texto(`strings`) y booleanos (`bool`), que solo pueden contener dos valores: verdadero o falso.

Python nos provee una función útil para chequear que tipo de variable contiene cada variable definida: `type()` Chequea la documentación (https://docs.python.org/3/library/functions.html?highlight=type#type).

Enteros (`int`)

El primer tipo de datos que podemos tener son numeros enteros, que corresponde al tipo `int` en Python.

In [ ]:
```
#Declara la variable mi_numero y asignale el numero 4
mi_numero = 4

#Declara la variable otro_numero y asignale el numero -1
otro_numero = -1

#Chequea que tipo de variables son mi_numero y otro_numero
print(type(mi_numero))
print(type(otro_numero))
```

In [ ]:
```
#Imprime el resultado de multiplicar mi_numero y otro_numero
print(mi_numero*otro_numero)
```

FLOTANTES (`float`)

Además de enteros, los datos también pueden ser de tipo flotantes. Esto es pueden contener numeros como `5.5`, `7.7`, `9014019401.43`. Este tipo de variable corresponde a `float` en Python.

In [ ]:
```
#Declara la variable float_numero y asignale el numero 4.9
float_numero = 4.9

#Chequea que tipo de variable es float_numero
print(type(float_numero))
```

In [ ]:
```
#Imprime el resultado de elevar float_numero a mi_numero, declarado anteriorm
ente
print(float_numero**mi_numero)
```

BOOLEANOS (`bool`)

Una variable booleana es una variable lógica que admite solo dos valores `True` y `False`.

In [ ]:
```
#Corre la siguiente linea
mi_bool = True
```

In [ ]:
```
#Ahora asigna a la variable otra_bool el valor False
otra_bool = False
```

In [ ]:
```
#Chequea que tipo de variables son mi_bool y otra_bool
print(type(mi_bool))
print(type(otra_bool))
```

Pese a que las variables booleanas son definidas con los valores `True` y `False`, Python las trata como numeros.

In [ ]:
```
#Prueba que ocurre si sumamos mi_bool y otra_bool
mi_bool + otra_bool
```

CADENA DE CARACTERES (`string`)

La cadena de caracteres o string es otro tipo de datos que podemos encontrar en Python. Estas variables poseen de particular que se definen utilizando comillas, como en el ejemplo siguiente:

In [ ]:
```
#Corre las siguientes lineas
mi_frase = "Aprender Python es muy sencillo"
segunda_frase = 'Solo hay que practicar'
print(mi_frase)
print(segunda_frase)
```

Comillas

Como habras observado arriba, se pueden utilizar comillas dobles como simples para definir strings. La regla es que utilices las mismas comillas para **abrir y cerrar** el string.

In [ ]:
```
#Define un string que quieras y asignaselo a la variable mi_primer_string
mi_primer_string = "Aca va un string"

#Imprime la variable mi_primer_string
print(mi_primer_string)
```

In [ ]:
```
#Define un segundo string y asignaselo a la variable segundo_string
segundo_string = "Aca va otro string"
```

In [ ]:
```
#Ahora aplica el operador suma (+) entre las dos variables y fijate que pasa
mi_primer_string + segundo_string
```

Como observaste los strings también pueden sumarse, o en terminos más correctos, **concatenarse**.

Python también nos permite acceder a un elemento del string o una porción de él. Esto se realiza mediante el uso de indices que se especifican con corchetes `[]`.

In [ ]:
```
#Defino un string
prueba_indice = "Acceder a un caracter es facil"

#Accedo al tercer caracter
prueba_indice[3]
```

¿Observaste bien? Si especifico el indice 3, me trae el 4to caracter. Esto es porque en Python los indices comienzan en 0. Entonces, el primer caracter tiene indice 0, el segundo 1 y asi sucesivamente.

In [ ]:
```
#Accede al octavo caracter del string prueba_indice
print(prueba_indice[7])

#Accede al quinto caracter del string prueba_indice
print(prueba_indice[4])
```

También es posible acceder a una porción del string. Esto se hace utilizando también corchetes [], pero se especifica el indice donde comienzo, luego : y luego el indice donde termino.

In [ ]:
```
#Corre la siguiente linea
print(prueba_indice[8:22])
```

¿Notaste algo en particular? Para poder traer la porción entre el 9no y el 22do caracter, debo especificar el caracter donde comienza (`8`) y la posición donde termina (`21`).
Sin embargo, especificamos 22. ¿Por qué? Porque Python **SIEMPRE excluye** la última posición que le especifiquemos. Entonces si especificamos hasta 21, traera hasta el 20. Si queremos hasta el 21 inclusive, debemos colocar 22 entonces.

In [ ]:
```
#asigna el string a la variable mi_frase_prueba
mi_frase_prueba = "Es esencial que leas mucho"
mi_segunda_prueba = "Para muchos era invisible aunque sus ojos se destacaban"
```

In [ ]:
```
#Selecciona el substring esencial de la variable mi_frase_prueba y asignalo a
la variable primera_palabra
primera_palabra = mi_frase_prueba[3:11]

#Selecciona el substring invisible de la variable mi_segunda_prueba y asignal
o a la variable segunda_palabra
segunda_palabra = mi_segunda_prueba[16:25]

#Selecciona el substring ojos de la variable mi_segunda_prueba y asignalo a l
a variable tercera_palabra
tercera_palabra = mi_segunda_prueba[37:41]
```

In [ ]:
```
#Concatena los strings "Lo", primera_palabra, "es", segunda_palabra, "a los", tercera_palabra.
#Imprime el resultado. No te olvides de especificar espacios usando el string " "
print("Lo"+" "+primera_palabra+" "+"es"+" "+segunda_palabra+" "+"a los"+" "+tercera_palabra)
```

In [ ]:
```
#Ahora concatena tercera_palabra y la variable mi_numero definida anteriormente
tercera_palabra + mi_numero
```

Como podrás ver no es posible concatenar variables de distintos tipos. Sin embargo, hay algunas funciones que nos brinda Python que nos permiten transformar entre tipos de variables.

- `str()` convierte a string una variable o valor

In [ ]:
```
#Aplica la función str a la variable mi_numero y guardala en la variable mi_numero_str
mi_numero_str = str(mi_numero)
```

In [ ]:
```
#Concatena tercera_palabra y mi_numero_str
tercera_palabra + mi_numero_str
```

- `int()` convierte a entero una variable o valor

In [ ]:
```
#Aplica la función int a mi_numero_str y multiplicalo por 5
int(mi_numero_str) * 5
```

Otras funciones similares son: `float()` (https://docs.python.org/3/library/functions.html#float) y `bool()` (https://docs.python.org/3/library/functions.html#bool).

Inmutables

Todos los tipos de variables que vimos hasta ahora son **inmutables**. Esto significa que no pueden ser alteradas después de haber sido creadas.

In [ ]:
```
#Corre la siguiente linea y observa que occurre
variable_inmutable = "Los strings son inmutables en Python"
variable_inmutable[8] = "f"
```

Lo que nos dice el error, es que una vez asignada una variable, no puedo cambiar parte de ella. Además de los mencionados hasta ahora, Python nos ofrece otras estructuras de datos que son más flexibles y permiten agrupar varios valores. Además muchas de ellas son mutables, es decir, permiten que asignemos o alteremos valores una vez definidas.

Operaciones aritméticas

Ahora que conocemos los cuatro tipos de dato básicos, veamos algunas operaciones que se pueden aplicar sobre ellos. Comencemos por los números enteros y de coma flotante. En la clase anterior estuvimos sumando dos enteros de la siguiente forma:

In [ ]:
```
#Suma
print(7+5)
#Resta
print(5-2)
#Multiplicacion
print(3 * 5)
#División
print(10 / 2)
```

Los objetos alrededor de un operador (en este ejemplo, 7 y 5) pueden ser simples números enteros o de coma flotante, o bien expresiones que retornen un número entero o de coma flotante, o bien **variables** que contengan un número entero o de coma flotante.

In [ ]:
```
a = 5
pi = 3.1415
print(a+7)
print(pi-pi)
print(pi*3)
print(a/pi)
```

Estas cuatro operaciones aritméticas retornan un entero cuando los dos objetos sobre los que actúan son enteros. Si alguno de ellos es un número de coma flotante, o bien ambos, el resultado es un número de coma flotante. De esto se sigue que pi - pi retorne 0.0 (un número de coma flotante) y no 0 (un número entero).

También podemos hacer uso de los paréntesis para agrupar expresiones, al igual que en matemática:

In [ ]:
```
print(7 + 5 * 3)
print((7 + 5) * 3)
```

Operaciones comparativas

Siguiendo con los números enteros y de coma flotante, además de las operaciones aritméticas disponemos de operaciones de comparación vía los siguientes operadores:

```
== (igualdad),
!= (desigualdad),
> (mayor),
>= (mayor o igual),
< (menor) y,
< (menor o igual).
```

Al ejecutar alguna de estas cinco operaciones, el resultado siempre es un `booleano` (es decir, True o False) según la comparación sea verdadera o falsa, así como el resultado en las operaciones aritméticas siempre era un número entero o de coma flotante. Ejemplos:

In [ ]:
```
print(7 > 5)
print(7 <= 5)
print(7 != 5)
print(7 == 5)
print(5 >= 7)
print(5 < 7)
```

Los operadores **>** , **>=** , **<** y **<=** solo pueden aplicarse sobre números enteros o de coma flotante; no obstante, los operadores de igualdad y desigualdad permiten comparar indistintamente cualquiera de los cuatro tipos de dato básicos (y también otros que veremos más adelante).

Ejercicios:

- Escribir un programa que muestre por pantalla la frase ¡Hola mundo!

- Escribir un programa que almacene la cadena ¡Hola Mundo! en una variable y luego muestre por pantalla el contenido de la variable.

- Escribir un programa que pregunte el nombre del usuario en la consola y después de que el usuario lo introduzca muestre por pantalla la cadena ¡Hola < nombre >!, donde < nombre > es el nombre que el usuario haya introducido.

- Escribir un programa que pregunte el nombre del usuario en la consola y un número entero e imprima por pantalla en líneas distintas el nombre del usuario tantas veces como el número introducido.

- Escribir un programa que pida al usuario su peso (en kg) y estatura (en metros), calcule el índice de masa corporal y lo almacene en una variable, y muestre por pantalla la frase Tu índice de masa corporal es < imc> donde < imc> es el índice de masa corporal calculado redondeado con dos decimales.

- Supongamos que tenemos $100, que podemos invertir obteniendo el 10\% de interes cada año. Luego del primer año, tendremos 100 x 1.1, luego de dos años tendremos 100 x 1.1 x 1.1. ¿Cuanto tendrás luego de 7 años?

- Mostrar el precio del IVA de un producto con un valor de 100 y su precio final

- Escribir un programa donde pida ingresar 3 notas y luego muestre por pantalla el promedio de ellas

- Modificar el programa anterior de modo que la nota final sea la suma del 15% de la primer nota, el 35% de la segunda y el 50% de la tercer nota

- Crear, en un script de Python, tres variables nombradas a, b y c con valores numéricos cualesquiera; una cuarta llamada resultado que sea la suma de las primeras tres, y por último imprimir en pantalla cada una de ellas. Antes de mostrar el valor de cada variable, indicar su nombre en una línea anterior