MANUAL PYTHON

Python 3.X: lenguaje de programación interpretado, de tipado dinámico y orientado a objetos (3.10 es la última versión de Python). https://www.python.org/

- **Lenguaje interpretado (vs. Compilado):** lenguaje de alto nivel que es convertido a lenguaje de máquina a medida que es ejecutado, sin necesidad de ser compilado. https://blog.makeitreal.camp/lenguajes-compilados-e-interpretados/
- **Tipado dinámico (vs. Estático):** un lenguaje de programación es dinámicamente tipado si una variable puede tomar valores de distinto tipo, es decir, cuando el tipo de dato de una variable puede cambiar en tiempo de ejecución. https://recursospython.com/guias-y-manuales/tipado-dinamico-y-tipado-fuerte/
- **Tipado fuerte (vs. Débil):** ante una operación entre dos tipos de datos incompatibles, arroja un error (durante la compilación o la ejecución, dependiendo de si se trata de un lenguaje compilado o interpretado) en lugar de convertir implícitamente alguno de los dos tipos. https://recursospython.com/guias-y-manuales/tipado-dinamico-y-tipado-fuerte/
- **Orientado a objetos (*Object Oriented Programming*, OOP):** la programación orientada a objetos es un modelo de programación informática que organiza el diseño de software en torno a datos u objetos, en lugar de funciones y lógica. Un objeto se puede definir como un campo de datos que tiene atributos y comportamiento únicos. https://www.computerweekly.com/es/definicion/Programacion-orientada-a-objetos-OOP

IDE (entorno de desarrollo integrado, Integrated Development Environment): es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. Normalmente, un IDE consiste en un editor de código fuente, herramientas de construcción automáticas y un depurador. *(ex. Anaconda, Visual Studio…)*
https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado

Tipos de variables en Python:

- **str** variable string: cadenas de texto, los caracteres siempre han de ir entre comillas (´simples´,”dobles” o ´”triples´”).
- **int** variable integer: números enteros (positivos y negativos), incluido 0.
- **float** variable float: números decimales (positivos y negativos).

- **bool** variable boolean: True or False, son binarias (T y F siempre en mayúscula).

Comandos Python:

Variable

- **variable = *expresión*** declarar una variable
- **variable = none** declarar una variable vacía
- **variable = input("*mensaje*")** ejecuta el código hasta llegar aquí, donde se detiene esperando a que el interlocutor responda por teclado. Todo lo que se introduzca será de tipo string.
- **variable = int(input("*mensaje*"))** para que interprete los que se introduzca como integer
- **variable = float(input("*mensaje*"))** para que interprete los que se introduzca como float

Comentarios

- **#** comentario de una línea
- **""" """** comentario de más de una línea

Print

- **print()** imprimir algo en la terminal
- **print(type())** saber el tipo de variable
- **print(int())** imprimir una variable como integer
- **print(type(str()))** imprimir el tipo de variable al que estás cambiando una variable (redundancia)
- **print("*mensaje*" + variable)** si la variable es string
- **print("*mensaje*" + str(variable))** si la variable no es string
- **print("*mensaje*", variable)** da igual el tipo de variable (opción más usada)
- **print("*mensaje* %s *mensaje* %f.2"%(variablestr,variablefloat))** una manera diferente de introducir variables en un mensaje impreso por pantalla. **%f.2** redondea el float a 2 decimales

Función

- **def función(parámetro*1*, parámetro*2*,…,parámetro*n*)**

  acción si queremos darle un valor fijo a uno de los parámetros, hay que colocar ese valor a la derecha: (par1, par2, par3=1)

- **función(parámetros)** llamar a la función

- **return(parámetros)** obtener un resultado sin que lo imprima en la terminal, puede devolver más de un valor
- **def función(parámetro_fijo, *arbitrario)**

  acción los arbitrarios son listas y tienen que ir a la derecha de los fijos. Poner 2 asteriscos (**) si es un diccionario
- **(lambda x, y, z: acción)(lista)** función anónima. No funcionan con bucles ni con return. Se pueden poner parámetros en vez de lista pero hay que sustituir los paréntesis por una coma.
- **list(map(función, lista))** devuelve una lista aplicando una función a cada elemento
- **import functools**

  functools.reduce(función, lista) devuelve un valor de la lista. Interesante para cuando queremos sumar todos los valores de una lista o concatenar
- **list(filter(función, lista))** devuelve una lista filtrada con la función

Listas

- **lista = []** crear lista (a.k.a. vectores, arrays)
- **lista = [0, 1, 2, 3…]** posiciones de una lista
- **lista = […-4, -3, -2, -1]** posiciones de una lista en negativo
- **print(lista[*posición en lista*])** imprimir un elemento de la lista
- **print(lista[*inicio:final:salto*]** si en inicio o en final no pone nada, se interpreta como inicio=-∞ o final=∞. En final siempre coge hasta la posición anterior
- **print(type(lista[*posición en lista*])** imprime el tipo de elemento
- **variable = lista[*posición en lista*]** variable = elemento de lista
- **variable = lista[*posición en lista*][*posición en sublista*]** si la posición en la lista está ocupada por una sublista. Esto vale para todas las funciones con lista
- **lista[*posición en lista*] = *expresión*** sustituye un elemento de la lista por lo que queramos
- **lista.append(*elemento*)** añadir un elemento a la lista
- **lista.extend(*elemento, elemento…*)** añadir varios elementos a la lista
- **lista.copy()** copiar una lista
- **lista.count(*elemento*)** número de veces que se repite ese elemento en la lista
- **len(lista)** número de elementos que contiene una lista
- **lista.index(*elemento*)** te dice la posición de ese elemento
- **lista.index(*elemento, inicio, fin*)** retorna la primera posición donde se encuentra el elemento
- **lista.insert(*posición, elemento*)** inserta ese elemento en esa posición

- **lista.clear()** vaciar una lista pero que siga existiendo
- **del lista** borrar una lista por completo
- **lista.pop(*posición*)** borrar el elemento de esa posición (nodo). Si no se indica posición borra el último elemento
- **lista.remove(*elemento*)** borra el primer nodo que contenga ese elemento
- **del lista[*posición*]** borrar el elemento de esa posición (nodo)
- **lista.sort()** ordena la lista de menor a mayor
- **lista.reverse()** invierte el orden de los elementos de la lista
- **max(lista)** hace referencia al máximo de la lista
- **min(lista)** hace referencia al mínimo de la lista

Condicionales

- **if *condición*:**

  Acción

  elif *condición:*

  Acción

  else:

  Acción
- **print("*mensaje*" if *condición* else "*mensaje*")** condicional en una sola línea
- **if variable in(*conjunto de valores)*** operador in
- **if type(variable) is str** operador is (se puede sustituir str por bool, float o int)

Estructuras repetitivas

- **while *condición:***

  Acción mientras se cumple la condición repite la acción
- **for variable in *estructura a recorrer:***

  Acción se crea una variable que va tomando como valor cada elemento de una estructura consecutivamente (lista, diccionario, string…)
- **for variable in range(0,10,2):**

  Acción la variable empieza siendo igual a 0, acaba siendo igual a 9 (siempre uno menos), y salta de 2 en 2 (var=0, var=2, var=4, var=6, var=8). Si solo pones un número interpreta que empieza en 0 y acaba en el número anterior. El range es una función para el for.
- **variable = 0**

while/for/for + range… variable < n

variable += 1 contador que se va acumulando para finalizar la estructura repetitiva

- **for variable in range(x,y):**

  print(variable, end=" ") para imprimir una secuencia en horizontal en vez de en vertical y con un espacio en medio (end=" ")

Cadenas

- **variable.lower()** convierte todas las letras de una cadena a minúscula
- **variable.swapcase()** convierte todas las mayúsculas de la cadena a minúsculas y viceversa
- **variable.title()** primera letra en mayúscula y el resto en minúscula
- **variable.startswith(*letras, inicio, final*)** comprueba si la cadena contiene las letras entre la posición inicial y final. La primera letra es 0 y la última posición es la anterior a final, igual que con las listas
- **variable.endswith(*letras, inicio, final*)** comprueba si la cadena termina con esas letras
- **variable = "*mensaje*" + "*mensaje*"** concatena
- **variable = "*mensaje*"*número** multiplica el mensaje por x número de veces
- **variable += "*mensaje*"** añade texto a cadena existente
- **variable.find("*elemento*")** te dice en qué posición empieza el elemento que buscas
- **variable.replace("*elemento a sustituir*","*elemento sustituto*")** sustituye elementos
- **len(variable)** longitud de una cadena (igual que con las listas)
- **strip()** elimina espacios iniciales y finales de una cadena. Si metes un elemento en el paréntesis y está al principio y al final también lo elimina.
- **rstrip()** elimina elemento del final
- **lstrip()** elimina elemento del principio
- **variable.split("*elemento de separación*", *número de trozos*)** convierte una cadena en una lista
- **"elemento entre las partes".join(lista)** convierte una lista en cadena
- **variable.isalpha()** devuelve True si hay al menos una cadena de caracteres y todos son letras, de lo contrario devuelve False

Diccionarios

- **clase = {}** crear un diccionario. Es una colección que relaciona una clave con un valor. En lugar de índice, se ordenan por la clave.
- **print(clase[clave])** imprime el valor asociado a esa clave
- **print(clave in clase)** True si la clave esta en el diccionario, False si no
- **clase[clave] = valor** añade elementos al diccionario
- **del clase[clave]** elimina elementos del diccionario
- **clase.clear()** vaciar un diccionario
- **del(clase)** eliminar un diccionario
- **sorted(clase)** ordena el diccionario por claves en una lista
- **sorted(clase, reverse=True)** ordena de manera inversa
- **from operator import itemgetter**

  print(sorted(compra.items(), key=itemgetter(1)))
  ordena el diccionario por valores (1 en itemgetter)
- **len(clase)** número de elementos de un diccionario
- **clase.keys()** hace referencia solo a las claves del diccionario
- **clase.values()** hace referencia solo a los valores del diccionario
- **for i in clase:** recorre todos los elementos del diccionario, se puede usar con clase.keys() y clase.values() también
- **for i, j in clase.items():**

  print(i,j) imprime todos los items (clave, valor)
- **for i,j in enumerate(clase):**

  print(i,j) imprime el índice de posición junto a su clave

<u>Ficheros</u>

- **fichero** python lee todo lo que hay en un archivo de texto como tipo string, independientemente de si son números o no. Para hacer operaciones con números de un archivo de texto, primero hay que castearlos: int() o float()
- **fichero = open(input())** pedir un fichero por teclado
- **fichero = open("*archivo*", "modo")**

  operación Abrir un archivo. Si el archivo está en la misma ubicación que el archivo.py de Python podemos poner directamente el nombre del archivo. Si está en una ubicación diferente hay que poner la ruta con una r delante o con doble //

  fichero.close()
- **with open("*archivo*", "modo") as fichero:**

operación abrir un archivo a partir de Python 2.5
(aquí no hay necesidad de cerrarlo)

- **modos:** formas de abrir un archivo

  r (lectura)

  w (escritura si existe el fichero)

  a (escritura, si existe el fichero lo añade al final)

  + (si pones el modo junto con + hace lectura y escritura)

  wb (escribe en binario)
- ***archivo*.closed** true si el archive está cerrado, false si está abierto
- ***archivo*.mode** devuelve el modo con el que se abrió el archivo
- ***archivo*.name** devuelve el nombre del archivo
- **fichero.read()** lee el archivo de principio a fin
- **fichero.read(*número de bytes*)** a partir de qué byte se lee hasta el final
- **fichero.readlines()** devuelve la siguiente línea
- **fichero.seek(*byte*)** mueve el puntero hacia el byte que le digamos
- **fichero.tell()** retorna la posición actual del puntero
- **fichero.write(*cadena*)** escribe cadena dentro del archivo
- **fichero.writelines(*secuencia*)** escribe línea a línea (tiene que ir con una estructura repetitiva)

Programación orientada a objetos

En python todos son objetos (strings, diccionarios, listas...) excepto los float y los integers. Los objetos tienen atributos (variables) y métodos (funciones). En python podemos declarar objetos y darles sus propios métodos y atributos:

- **class clase (object)** crear una clase. Los paréntesis y el object (nombre de la clase de la que hereda) solo son necesarios en caso de que queramos que la clase herede de otra clase.
- **class clase (object1, object2,...)** herencia múltiple. Crear una clase que herede de varias clases.
- **object.init(self, atributos)** si la clase hereda de otra clase (object), hay que decirle en el paréntesis los atributos que quiero que herede.
- **def __init__ (self, atributos)** método mágico para hacer el contructor de **self.atributo = atributo** clase. Si el atributo no tiene valor inicial, se sobreescribe como en el ejemplo.
- **def __str__ (self)** método mágico para mostrar por pantalla los valores **print(self.atributo)** de los atributos.

- **variable = clase(parámetros)** crear una nueva instancia de una clase. Los parámetros en el paréntesis serán introducidos en el constructor, son valores para los atributos.
- **def método (self, parámetros)** crear nuestro propio método
- **variable.método()** ejecutar un método
- **variable.atributo** devuelve el valor del atributo en la instancia "variable"
- **__getattr__** método mágico para acceder a un atributo
- **print(getattr(variable, "atributo")** imprimir el valor de un atributo en la instancia "variable"
- **variable.__setattr__(atributo, nuevo valor)** método mágico para modificar un atributo
- **__delattr__** método mágico para eliminar un atributo

Librerías

- **import *librería*** importar una librería
- **import math**
- **import random**
    - **random.randint(x,y)** generar un número entero aleatorio entre x e y

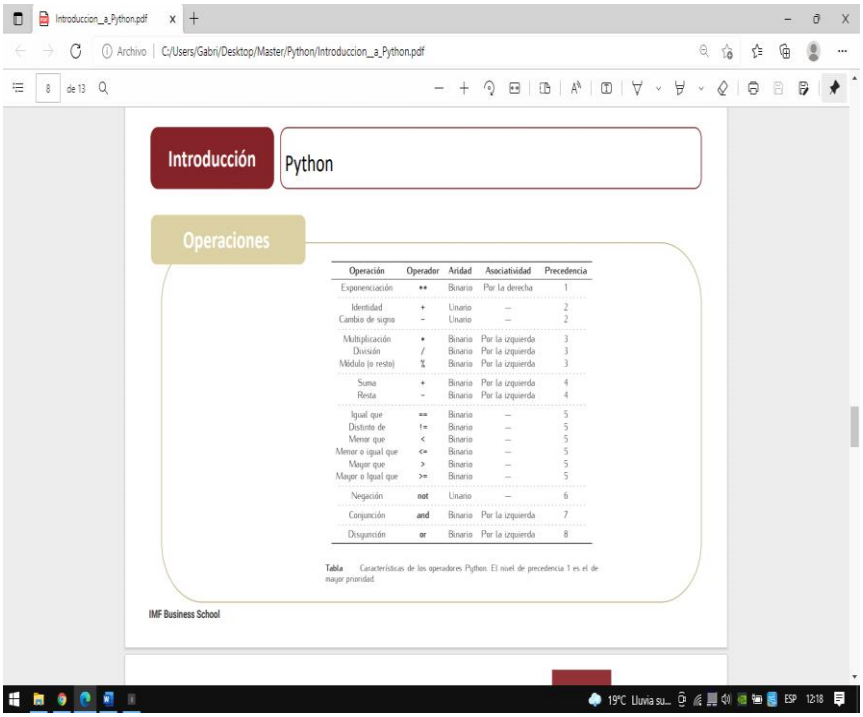
Palabras reservadas: no se pueden utilizar para declarar una variable o una colección.

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

Funciones predefinidas: no se necesita importar ninguna librería.

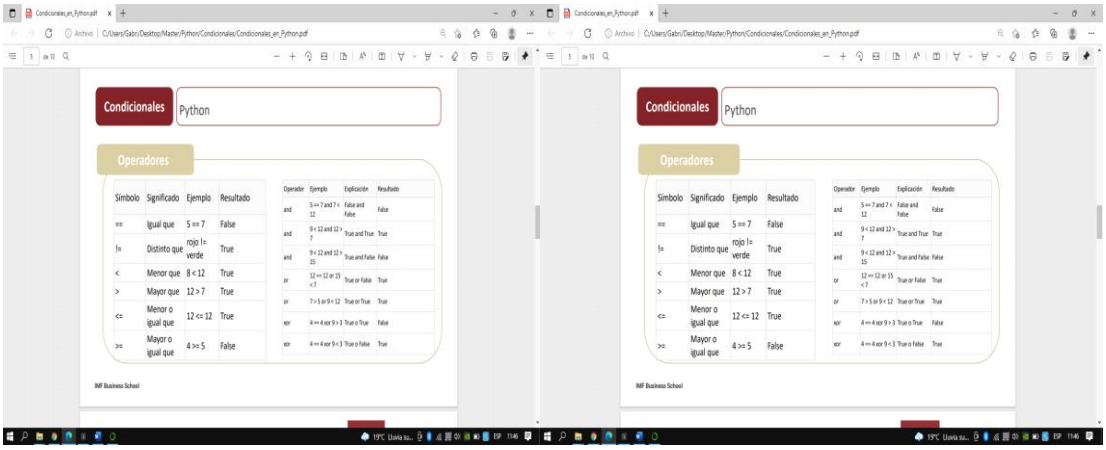
- **abs():** calcula el valor absoluto de un número.
- **float():** convierte un número o cadena numérica a flotante.
- **int():** convierte un número o cadena numérica entera a entero.
- **Str():** convierte un numero en una cadena.
- **round():** redondeo. Puede usarse con uno o dos argumentos

Operaciones:



variable//número *calcula el cociente (número entero)*

- **comparadores para los condicionales**



- **import math:** operaciones para las que hay que importar la librería math.