UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

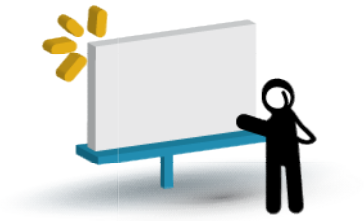
UNIDAD DIDÁCTICA II

DIPLOMATURA EN PYTHON

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo I – Nivel Inicial I

Unidad II – Tipos de datos I.



Presentación:

Python cuenta con diferentes tipos de datos, desde simples enteros que podemos trabajar como si se tratara de una calculadora, hasta números complejos o fraccionarios que podríamos utilizar en trabajos de electrónica o convergencia de series. En esta unidad comenzaremos a describir los tipos de datos con los cuales podemos contar y presentaremos la herramienta de trabajo IPython, la cual es muy útil para comenzar a conocer el lenguaje.

Objetivos:

Que los participantes:

Conozcan la herramienta IPython.

Comiencen a conocer los distintos tipos de datos que podemos trabajar en python.

Bloques temáticos:

1.- Jupyter Lab.

2.- Números.

3.- Strings.

4.- Listas.

5.- Diccionarios.

6.- Herramientas de sistema.

Consignas para el aprendizaje colaborativo

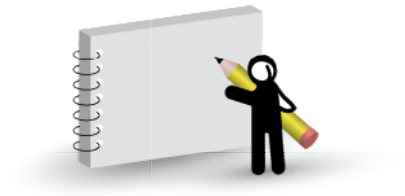
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Jupyter Lab

Para comenzar a trabajar en esta unidad, presentaremos ***una herramienta muy interesante la cual le añade funcionalidades extras al modo interactivo incluido en python***, como resaltado de línea y autocompletado de datos: Jupyter Lab.

Para instalarlo seguiremos unos pasos sencillos.

Paso 1 – actualizamos pip

Python posee un repositorio de paquetes extras desarrollados por terceros, que le añaden funcionalidad a nuestras aplicaciones. Estos componentes pueden ser instalados y administrados mediante el gestor de paquetes pip, y antes de avanzar nos aseguraremos de poseer la última versión.

Si estamos en Windows, abrimos un cmd como administrador y ejecutamos.

```
pip install --upgrade pip
```

Nota: Recordar que en Linux o Mac en lugar de pip es pip3

Paso 2 - Instalamos Jupyter

```
pip install jupyterlab
```

Paso 3 – Ejecutamos el organizador de trabajos

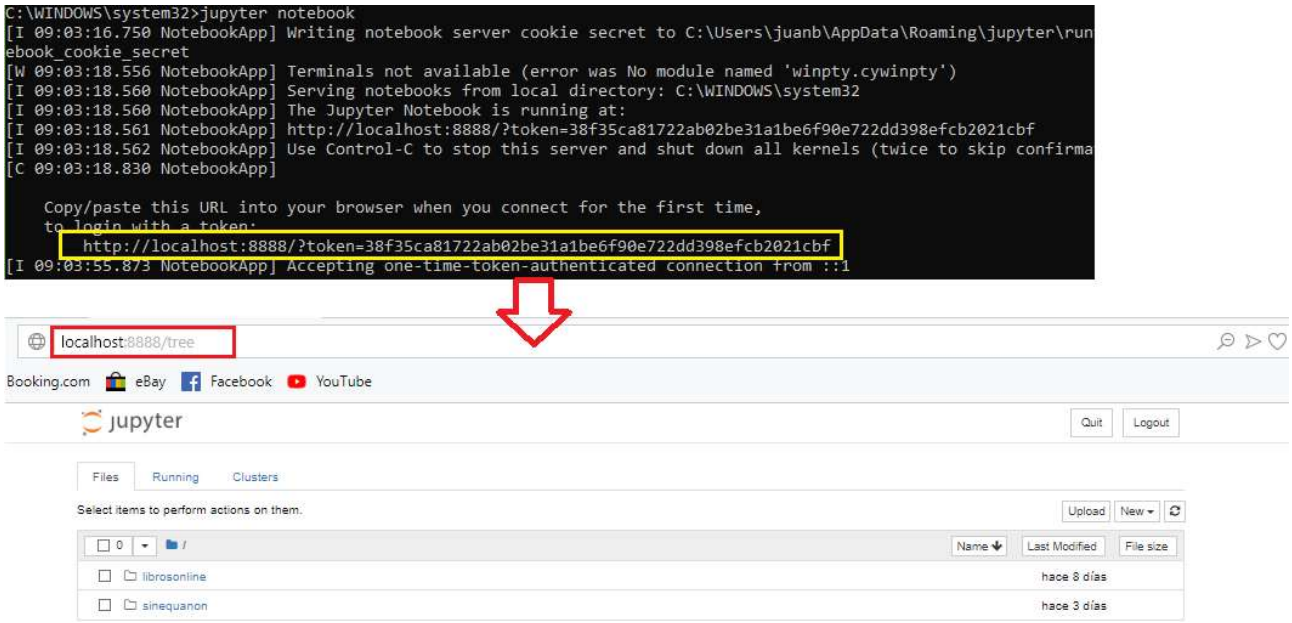
Ingresamos a nuestro directorio de trabajo desde el cmd, por ejemplo el escritorio (en mi caso mi usuario es juan)

```
C:\WINDOWS\system32>cd /
C:\>cd users
C:\>cd juan
C:\juan>cd desktop
C:\juan\desktop>
```

Y ejecutamos:

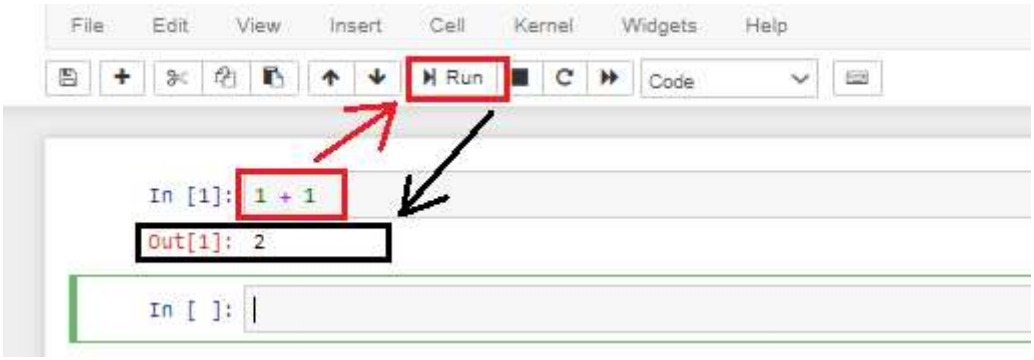
jupyter lab

Observamos como en el cmd nos aparece la indicación de que se está ejecutando el servidor local en el puerto 8888 e inmediatamente se abre el explorador por defecto mostrando el tablero de trabajo.

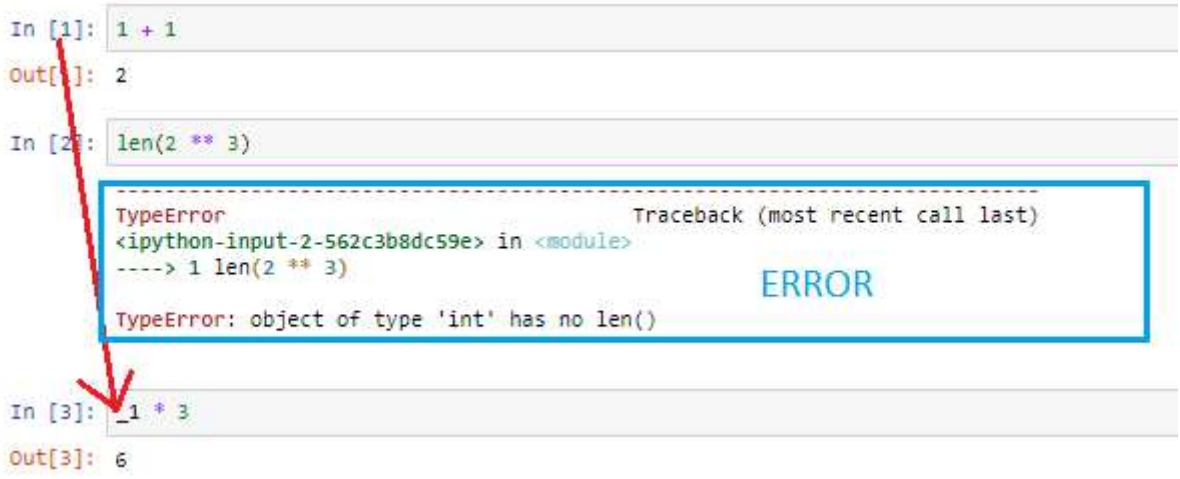


Paso 4 – Creamos un nuevo nootebok

Creamos un nuevo notebook de Ipython presionando en new, los notebooks se organizan en celdas, y si escribimos en una celda y ejecutamos nos retorna el resultado agregando una nueva celda.



Es interesante utilizar IPython en lugar del cmd por la gran cantidad de funcionalidades adicionales que podemos utilizar, por ejemplo un guión bajo, nos permite recuperar el resultado de una celda escrita anteriormente, mientras que si cometemos un error nos sale un mensaje preciso de la causa del error. En la siguiente imagen podemos ver como la línea 2 nos da un error, y como podemos referenciar a la línea 1 dentro de una línea de código ejecutable para utilizar un resultado previo.



2.- Números

Python soporta números enteros, de punto flotante o complejos, básicamente el intérprete actúa como una calculadora, utilizando operadores de forma análoga a como lo haríamos en C, en donde tenemos la posibilidad de utilizar paréntesis para agrupar términos, pero teniendo en cuenta que el uso de paréntesis puede alterar el resultado. Veamos un ejemplo:

```
valor1 = 5 - 2 *6
valor2= (5 - 2) *6
print(valor1)
print(type(valor1))
print(valor2)
print(type(valor2))
```

Resultado

```
-7

<class 'int'>

18

<class 'int'>
```

Como vemos ambos resultados son del tipo entero ya que los números utilizados son del tipo entero. Esto no se cumple cuando utilizamos el operador de división el cual siempre nos retorna un número flotante:

```
valor3 = 10/2
print(valor3)
print(type(valor3))
```

Resultado

```
5

<class 'float'>
```

Si en lugar de dividir utilizando el slash "/" utilizamos un doble slash "//" obtenemos siempre la parte entera de una división, python descarta la parte fraccional y se queda con el entero. En este caso el valor lo entiende como entero y no como flotante.

También es posible trabajar con números octales o hexadecimales anteponiendo al número "0o" o "0x" respectivamente:

```
valor4 = 0x17
valor5= 0o17
print(valor4)
print(type(valor4))
print(valor5)
print(type(valor5))
```

Resultado

```
23

<class 'int'>

15

<class 'int'>
```

Notar que en ambos casos los considera como enteros.

Otros tipos de datos numéricos que podríamos utilizar son los decimales, las fracciones, los números booleanos o los números complejos, estos últimos son muy utilizados en electrónica.

3. Strings.

Los strings son una colección ordenada de caracteres utilizada para almacenar y representar información de bits y texto. Este tipo de datos se representan entre comillas simples o dobles, un ejemplo simple podría ser la declaración del número cinco como sigue:

```
mi_string = "5"
```

Es importante destacar que existe diferencia entre el cinco con comillas y sin comillas, en el primer caso se trataría de un string mientras que en el segundo caso python lo entendería como un entero. Por lo que si ejecuto:

```
print(5 *  “5”)
```

El resultado no sería el número entero 25, sino que daría (55555), el cual tampoco sería un entero, sino que python entiende del código anterior que lo que estamos queriendo hacer es repetir el string 5, cinco veces. Para confirmar esto podemos ejecutar el método type():

```
print(type(5 *  “5”))
```

El cual nos retorna el tipo de dato que representa (“5” * 5):

```
<class 'str'>
```

Como podemos ver nos dice que el resultado es del tipo ‘str’ lo cual equivale a un string.

Si quisiéramos que el resultado fuera convertido a un entero, podemos utilizar el método int():

```
print(type(int(5 *  “5”)))
```

Ahora el método type nos indica que el resultado es de tipo entero (int):

```
<class 'int'>
```

El siguiente cuadro nos presenta algunas de las operaciones que podemos realizar con strings y es una excelente referencia abreviada que podemos consultar conforme avanza el curso.

Operación	Interpretación
s = ' '	String vacío
s = " Hola"	Las dobles comillas se usan igual que las simples
s = 's\np\ta'	Secuencia de escape \n es salto de línea y \t tabulación
s = b'abc'	Si al string lo precede una b se trata de información guardada como bits
s = """ Comentarios	Las triples comillas definen bloque de comentario

Múltiples """	
s1 + s2	Concatenar Strings
s * 3	Repite un string
s[i]	Elemento i comenzando desde cero desde la izquierda
s[-i]	Elemento i comenzando desde uno desde la derecha
s[i:j]	Rango de i a j sin incluir j
s[i:]	Rango a partir de i
s[i:-1]	Rango a partir de i, sin incluir el último elemento
s[:]	Selecciona todos los elementos
s[i : j : k]	De i a j de a k
s[: :-1]	Lee S de derecha a izquierda
s[i : j :-1]	Desde j a i (orden inverso)
len(s)	Longitud
str(33)	Convierte a string '33'
ord('s')	Pasa a Unicode
chr(115)	Obtiene carecter a partir de Unicode
"a %s ellos" % todos	Da formato a expresión
	s = string, c = caracter, d=decimal, i=entero
	f = flotante decimal,
"a {0} ellos" .format(todos)	Da formato por posición en python 2.6, 2.7 y 3.x
"a {valor} ellos" .format(valor = 1)	Da formato por asignación
s.find('pa')	Búsqueda de pa dentro de S Retorna ubicación de primer caracter
s.rstrip()	Remueve espacio
s.replace('pa', 'xx')	Reemplaza pa por xx
s.split(' , ')	Separar según delimitador indicado
s.isdigit()	Test de contenido
s.lower()	Convierte S a minúscula
delim.join(s)	Crea un string con los elementos de S separados por delim
' '.join(s)	Crea un string con los elementos de S separados por espacio en blanco
for x in s: print(x)	Iteración
'p' in perro	Se fija si existe la letra p en el string perro
chars = list('manzana')	chars = ['m', 'a', 'n', 'z', 'a', 'n', 'a']
chars.append('r')	chars = ['m', 'a', 'n', 'z', 'a', 'n', 'a', 'r']
' '.join(chars)	'manzanar'

3.2. Inmutabilidad de los strings.

El objeto al cual apunta "s" no se ve modificado con ninguna de las operaciones realizadas, cada operación de Strings produce un nuevo objeto String dado que son inmutables en Python (no pueden ser modificados). De esta forma dado un string j = “Juan”, no se podría realizar por ejemplo lo siguiente:

```
j[0] = 'P'
```

Esta acción daría un error ya que los strings son inmutables. Sin embargo si podemos crear un nuevo objeto y asignarle el mismo nombre que el anterior:

```
j = j[1:] + 'w'
```

3.3. Métodos

Python posee una serie de palabras reservadas que al agregarlas a un string dado, mediante notación de punto (es decir el nombre del string + . + palabra reservada) nos permite realizar alguna operación como podría ser contar cuantas letras posee un determinado string. Estas palabras reservadas en realidad son funciones (métodos) de los cuales hablaremos más adelante y que internamente ejecutan una rutina que permite ejecutar una determinada acción (más adelante veremos cómo generar nuestras propias funciones, por ahora basta con considerarlas como cajas negras de las cuales no conocemos el script que poseen pero sabemos que al aplicarlas a un string realizan una determinada operación). La tabla presentada en esta sección recoge algunas de estas funciones o métodos, como ejemplo si tuviéramos el string S = “Hola” y ejecutáramos el código S.find('la') , podríamos determinar en qué lugar del string Hola se encuentra el subestring 'la'.

Para conocer todos los métodos aplicables a un string, podemos agarrar un string y escribir **dir(string)**, por ejemplo con “s” sería dir(s). Para saber cómo funcionan se utiliza el método **help**, de esta forma si queremos determinar cómo se utiliza el método “find”, podemos escribir:

```
help(s.find)
```

Y nos saldría una descripción de cómo funciona find.

Nota: Los métodos pueden concatenarse.

4. Listas.

Las listas son una enumeración de objetos de python agrupados entre corchetes y separados por coma, funcionan como los arrays en otros lenguajes y son NO INMUTABLES (pueden ser modificadas en el lugar) y pueden responder a todo tipo de secuencia de operaciones utilizada con strings. Debido a que son MUTABLES, soportan otras operaciones que los strings no, como las operaciones de asignación por índice que cambian las listas en el lugar.

4.1. Operaciones con listas.

El siguiente cuadro recopila algunas de las operaciones que se suelen utilizar con listas.

Operación	Interpretación
lista = []	Definición de lista vacía
lista = [False]*10	Lista con 10 elementos FALSE
lista = [123, 'abc', {}]	Lista con tres elementos
lista = ['Juan', 38, ['pintor', '5000']]	Listas de más de una dimensión
lista = list('pera')	Genera una lista con las letras de pera L = ['p', 'e', 'r', 'a']
lista =[i:j]	Longitud
len(lista)	Cantidad de elementos de la lista
lista 1 + lista 2	Suma dos lista
lista * 2	Genera una lista que posee los elementos
	de L repetidos dos veces
for x in lista: print(x)	Recorre e imprime cada elementos de L
lista.append(3)	Le agrega a la lista el número 3
lista.extend([1, 2])	Le agrega a L más de un elemento de lista
del lista [i]	Elemento
del lista [i:j]	Rango
lista [i] = 3	Asignar valor a componente i
lista [i:j] = [4, 5]	Asigno valor a componente i:j
print(lista [0])	Imprimir el primer elemento desde la izquierda
print(lista [-1])	Imprimir el primer elemento desde la derecha
lista =[[1,2][3,4]]	Matriz de 2*2
print(lista[0].split()[-2])	Juan

Ejemplos de reasignación en listas

Supongamos que tenemos una lista lista1 que apunta al objeto [0, 1, 2] si referenciamos a la lista desde lista2 y modificamos el objeto al cual hace referencia lista1, por ejemplo:

```
lista1 = 3
```

En este caso F2 sigue siendo igual a [0, 1, 2] y por lo tanto apuntaría a un objeto diferente, sin embargo si F1 solo cambiará el valor de uno de sus componentes, por ejemplo:

```
lista1[0] = 1
```

En este caso F2 pasaría a ser [1, 1, 2]; esto es porque en realidad no hemos cambiado en este último caso el objeto al cual referencia F1, sino que hemos alterado su valor. Los tipos de objetos sobre los cuales se pueden realizar este tipo de modificaciones se llaman “mutables”. Como ejemplo tenemos las listas y los diccionarios (de estos últimos hablaremos más adelante), por otro lado los objetos inmutables no permiten esta modificación, por ejemplo los números, los strings, las tuplas y los sets.

```
# ########################################
# Crear Lista
# ########################################
lista1 = [0, 1, 2]
lista2 = lista1                          # lista1 y lista2 son el mismo objeto
print(lista2)
# ########################################
# Renombramos primer lista
# ########################################
lista1[0]=4                              # lista1 y lista2 son el mismo objeto
print(lista2)
# ########################################
# Renombramos primer lista        # F1 y F2 son distintos objetos
# ########################################
lista1 = [2, 1, 2]
print(lista2)
input()
```

Ejemplos de reasignación en listas pero copiando

Si en lugar de reasignar creamos una copia del elemento original, al modificar el elemento original, el segundo elemento que hace referencia a la copia no se ve alterado.

```
import copy
# ########################################
# Crear Lista
# ########################################
lista1 = [0, 1, 2]
lista2 = copy.copy(lista1)                          # lista1 y lista2 son distintos
objetos
print(lista2)

input()
```

De string a lista

Si quisiera modificar un string, podría convertirlo en una lista, modificar la lista y luego convertir la lista modificada a un string:

```
mi_string = 'Juan'            # mi_string es un string
lista = list(mi_string)        #Transformo mi_string en una lista
print(lista)                  #imprimo la lista
lista[0] = 'P'                #LA LISTA PUEDE SER MODIFICADA
palabra = ''.join(lista)  # Uno los elementos con un delimitador vacío
print(palabra)
print(type(palabra))
```

Resultado

```
['J', 'u', 'a', 'n']

Puan

<class 'str'>
```

Métodos de listas (append – pop – extend – remove – insert)

Al igual que con los strings, también con las listas tenemos una serie de términos que podemos utilizar, los cuales nos ayudan en casos concretos. Veamos algunos de los más conocidos operando sobre una lista en la cual tenemos guardada los títulos de algunas películas:

recorrerListas.py

```
peliculas = ["película1", "película2", "película3"]
print(peliculas)
print(peliculas[0])
print(len(peliculas))
# Con append agregamos datos al final de una lista
peliculas.append("película4")
print(peliculas)
# Con pop removemos el último dato de una lista
peliculas.pop()
print(peliculas)
# Con extend agregamos un grupo de datos al final de la lista
peliculas.extend(["película4", "película5"])
print(peliculas)
# Con remove removemos un dato específico de la lista
peliculas.remove("película4")
print(peliculas)
# Con insert insertamos un dato en un lugar específico de la lista
peliculas.insert(0, "película0")
print(peliculas)
```

Retorna.

```
['película1', 'película2', 'película3']
película1
3
['película1', 'película2', 'película3', 'película4']
['película1', 'película2', 'película3']
['película1', 'película2', 'película3', 'película4', 'película5']
['película1', 'película2', 'película3', 'película5']
['película0', 'película1', 'película2', 'película3', 'película5']
```

5. Diccionarios

Junto con las listas, los diccionarios son uno de los tipos de objetos incorporados por python, que presentan una gran flexibilidad. Mientras que podemos pensar las listas como una colección ordenada de objetos, se puede pensar en los diccionarios como colecciones desordenadas, que poseen como principal distinción el hecho de que los elementos se almacenan por clave, en lugar de por su posición. Mientras que las listas se pueden considerar como los *arrays* de otros lenguajes, los diccionarios toman el lugar de los registros, tablas de búsqueda, y cualquier otro tipo de registro, en donde los nombres de los elementos son más significativos que sus posiciones.

La siguiente tabla recoge un resumen de las operaciones más utilizadas.

Operación	Interpretación
d = { }	Diccionario vacío
d = { 'nombre: 'Juan', 'edad' : 39}	Dos ítems del diccionario
d= {'identificación': {'nombre': 'Juan'}}	Anidamiento
d = dict('nombre: 'Juan', 'edad' : 39)	Estructura alternativa
d = dict([('nombre', 'Juan'),('edad', 39)])	Pares clave/valor
d = dict(zip(['clave1','clave2'],['valor1', 'valor2']))	Crear lista con zip
d['identificación']['nombre']	Indexado por identificación y nombre
'edad' in d	Chequear si un campo se encuentra en el diccionario
d.keys()	Todas las claves
d.values()	todos los valores
d.items()	Tuplas de todas las (clave, valor)
d.copy()	copiar
d.clear()	remueve todos los items
d.update(d2)	Agrego d2 a d
d.get(clave)	Obtiene el valor de una clave
d.pop(key)	Borra un elemento por su clave y retorna el valor
list(d.keys())	Crea una lista que contiene las claves del diccionario
list(d.values())	Crea una lista que contiene los valores del diccionario
list(d.items())	Crea una lista que contiene las claves y los valores del diccionario
len(d)	Longitud

d[clave] = 42	Agrega una nueva clave con su valor

5. Herramientas de sistema.

Python posee varias herramientas para trabajar con nuestro sistema operativo y dos módulos especiales para poder trabajar con las herramientas, estos son “sys” y “os”. Para poder utilizarlos primero hay que importarlos:

```
import sys, os
```

Y luego podemos analizar cuántos componentes posee cada uno y cuáles son, agregando las siguientes líneas:

```
import sys, os

print(len(dir(sys)))
print(len(dir(os)))

print(dir(sys))
print(dir(os))
```

Sys

Sys tiene componentes relacionados con el intérprete de Python (ejemplo: path), nos permite obtener información del sistema, por ejemplo veamos el siguiente código:

```
# Imprime plataforma
print(sys.platform)
# El mayor entero dimensionado de forma nativa en la máquina
print(sys.maxsize)
# Versión del sistema
print(sys.version)
# Directorios donde busca los módulos en el orden en el cual busca
print(sys.path)
```

Analicemos la salida de lo ejecutado en la máquina que estoy usando en este momento.

```
win32

2147483647

3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)]

['C:\\Users\\juanb\\Desktop', 'C:\\Python37\\python37.zip', 'C:\\Python37\\DLLs',
'C:\\Python37\\lib', 'C:\\Python37', 'C:\\Python37\\lib\\site-packages', 'C:\\Python37\\lib\\site-
packages\\win32', 'C:\\Python37\\lib\\site-packages\\win32\\lib', 'C:\\Python37\\lib\\site-
packages\\Pythonwin']
```

Notemos que el primer lugar en donde está buscando es en el escritorio del usuario actual y que recién en sexto lugar se encuentra el directorio “site-packages” en donde se van a instalar los programas de terceros. Algo muy interesante es que podemos modificar incluso las variables de entorno del sistema operativo mientras el programa que lo esté realizando se encuentre en ejecución.

Asimismo la ruta dada por sys.path puede ser modificada utilizando:

- append
- insert
- pop
- remove

Esta forma de modificar el sys.path no es permanente, sino que existe sólo mientras dura la existencia del script.

OS

Está compuesto por variables y funciones que mapean el sistema operativo en el que corre Python. A modo de ejemplo, el siguiente script retorna un número de identificación del script que se está ejecutando (id de proceso ó process ID) y el directorio en donde se está ejecutando el script actual.

```
print(os.getpid())
print(os.getcwd())
```

Si quisiéramos modificar el directorio por uno determinado podríamos ejecutar: os.chdir().