



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

**Centro de
e-Learning**

UNIDAD DIDÁCTICA II
DIPLOMATURA EN PYTHON

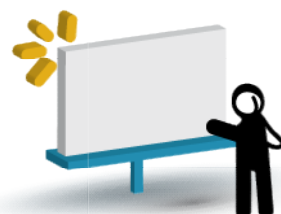
Centro de e-Learning SCEU UTN - BA.

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.

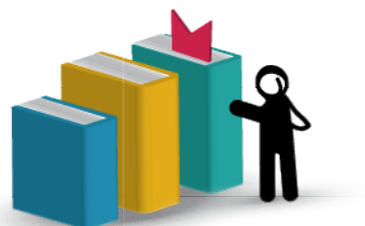


Objetivos:

Que los participantes:

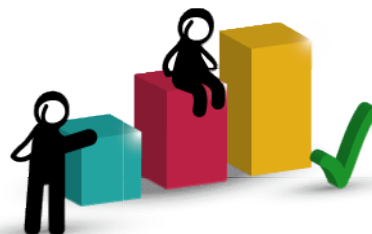
Conozcan la herramienta IPython.

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



Bloques temáticos:

- 1.- Jupyter IPython.
- 2.- Números.
- 3.- Strings.
- 4.- Listas.
- 5.- Diccionarios.
- 6.- Herramientas de sistema.



Consignas para el aprendizaje colaborativo

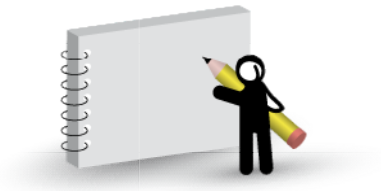
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- 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 IPython

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 IPython.

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.

OPCIÓN A

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

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

OPCIÓN B

También podemos instalarlo descargando el archivo de instalación desde:

<https://pip.pypa.io/en/stable/installing/>

Esta opción es válida también para cuando pip no está instalado

Una vez que lo descargamos por ejemplo en el Escritorio, ingresamos a un cmd y nos dirigimos al Escritorio.



¿Cómo hacemos esto?

Primero: Abrimos el cmd

El cual se ve en mi caso así:

```
Símbolo del sistema

Microsoft Windows [Versión 10.0.18362.836]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\juanb>
```

Esto dice que estoy parado dentro del directorio “juanb” que es mi usuario, dentro del directorio “Users” dentro del directorio principal “C”.

¿Cómo llego hasta el escritorio donde lo descargue?

Para ingresar en un directorio se ejecuta:

```
cd nombre_directorio
```

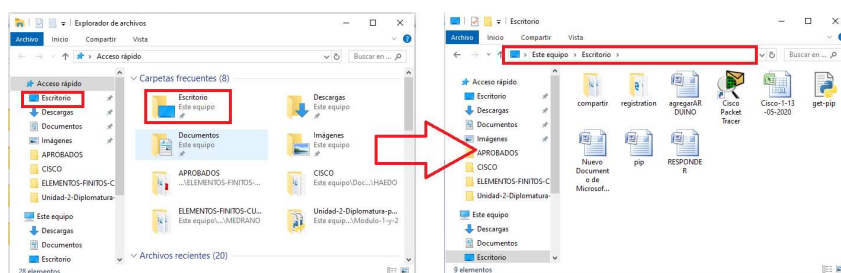
Para salir de un directorio se ejecuta:

```
cd ../
```

Para listar el contenido de un directorio ejecutamos:

```
dir
```

Entonces en mi caso me fijo donde estoy parado, abro una carpeta cualquiera y voy al escritorio:



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

Me paro en el buscador y se transforma el contenido mostrándome la ubicación desde el disco principal, el cual en mi caso es el “C”



Si solo me muestra un nombre, es porque el “Escritorio” está dentro del usuario actual, en otro caso nos mostraría algo como:

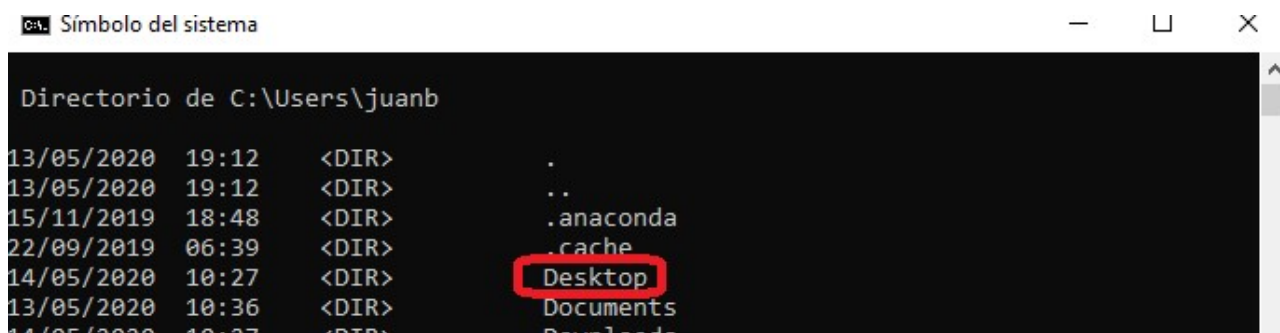


¿Ahora qué hago con esta información?

Bueno, aquí ya sé dos cosas:

- Que el directorio “**Escritorio**” internamente Windows lo llama “**Desktop**”
- Que el **Desktop** se encuentra dentro de mi usuario

Con esta información si ahora abro el cmd y hago un “dir” puedo ver que efectivamente el directorio “**Desktop**” se encuentra dentro





Ya lo ubique y ahora ¿Qué hago?

Pues puedo ingresar al directorio escribiendo:

```
cd Desktop
```

```
C:\Users\juanb>cd desktop  
C:\Users\juanb\Desktop>
```

Notar que puse “Desktop” y no “Escritorio”

Ahora puedo hacer un dir nuevamente y ubicar el archivo descargado:

```
C:\Users\juanb\Desktop>dir  
El volumen de la unidad C es Windows  
El número de serie del volumen es: 1C81-5C78  
  
Directorio de C:\Users\juanb\Desktop  
  
14/05/2020  10:27    <DIR>          .  
14/05/2020  10:27    <DIR>          ..  
09/05/2020  18:33             15.341  agregarARDUINO.docx  
13/05/2020  19:12             1.327  Cisco Packet Tracer.lnk  
13/05/2020  21:40          1.362.154  Cisco-1-13-05-2020.xlsx  
07/05/2020  21:18    <DIR>          compartir  
14/05/2020  10:27          1.868.059  get-pip.py  
14/05/2020  08:09          10.170  Nuevo Documento de Microsoft Office Word.docx  
14/05/2020  18:18
```

Ok ya lo ubique, y ahora ¿qué?

Ahor a solo resta ejecutarlo con:

```
python get-pip.py
```

LISTO, tendría que dejarme con la última versión de pip instalada.

Paso 2 - Instalamos Jupyter

```
pip install jupyter
```

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)

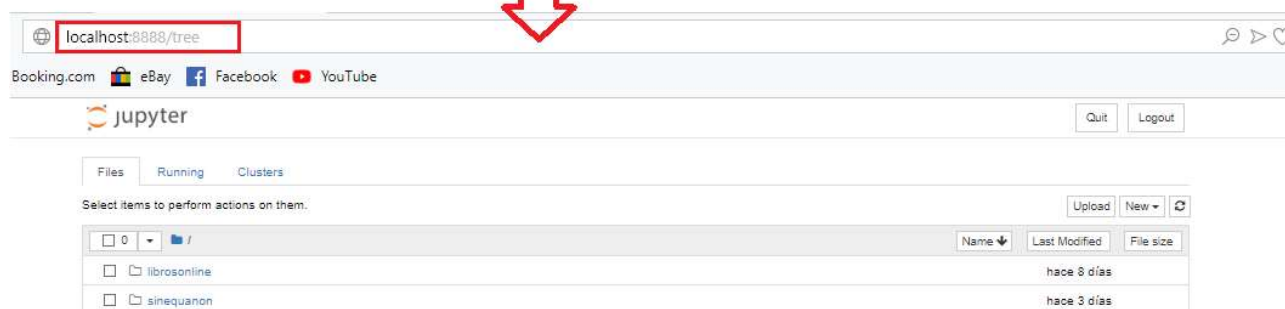
```
C:\WINDOWS\system32>cd /  
C:\>cd users  
C:\>cd juan  
C:\juan>cd desktop  
C:\juan\desktop>
```

Y ejecutamos:

```
jupyter notebook
```

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.

```
C:\WINDOWS\system32>jupyter notebook  
[I 09:03:16.750 NotebookApp] Writing notebook server cookie secret to C:\Users\juan\AppData\Roaming\jupyter\notebook_cookie_secret  
[W 09:03:18.556 NotebookApp] Terminals not available (error was No module named 'winpty.cwinpty')  
[I 09:03:18.560 NotebookApp] Serving notebooks from local directory: C:\WINDOWS\system32  
[I 09:03:18.560 NotebookApp] The Jupyter Notebook is running at:  
[I 09:03:18.561 NotebookApp] http://localhost:8888/?token=38f35ca81722ab02be31a1be6f90e722dd398efcb2021cbf  
[I 09:03:18.562 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirma  
[C 09:03:18.830 NotebookApp]  
  
Copy/paste this URL into your browser when you connect for the first time,  
to login with a token:  
http://localhost:8888/?token=38f35ca81722ab02be31a1be6f90e722dd398efcb2021cbf  
[I 09:03:55.873 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

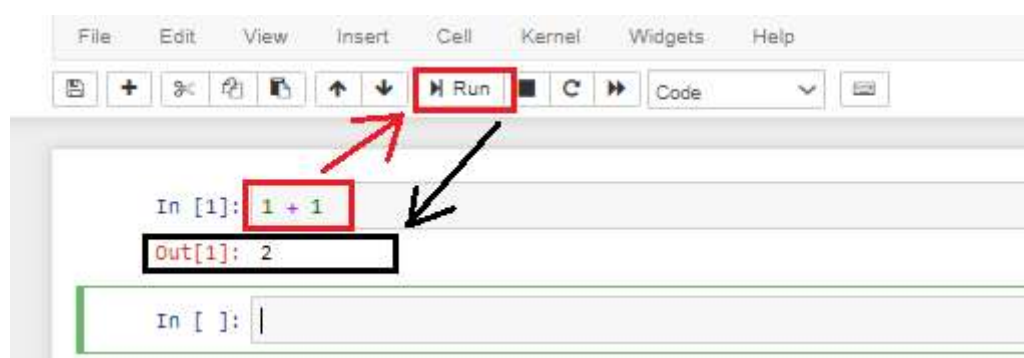


Centro de e-Learning SCEU UTN - BA.

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

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 gui3n 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.



```
In [1]: 1 + 1
```

```
Out[1]: 2
```

```
In [2]: len(2 ** 3)
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-2-562c3b8dc59e> in <module>
----> 1 len(2 ** 3)

TypeError: object of type 'int' has no len()
```

ERROR

```
In [3]: _1 * 3
```

```
Out[3]: 6
```

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'>
```

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



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
<code>S = ''</code>	String vacío
<code>S = " Hola"</code>	Las dobles comillas se usan igual que las simples
<code>S = 's\np\ta'</code>	Secuencia de escape <code>\n</code> es salto de línea y <code>\t</code> tabulación
<code>S = b'abc'</code>	Si al string lo precede una <code>b</code> se trata de información guardada como bits
<code>S = """ Comentarios Múltiples """</code>	Las triples comillas definen bloque de comentario
<code>S1 + S2</code>	Concatenar Strings
<code>S * 3</code>	Repite un string
<code>S[i]</code>	Elemento <code>i</code> comenzando desde cero desde la izquierda
<code>S[-i]</code>	Elemento <code>i</code> comenzando desde uno desde la derecha
<code>S[i:j]</code>	Rango de <code>i</code> a <code>j</code> sin incluir <code>j</code>
<code>S[i:]</code>	Rango a partir de <code>i</code>
<code>S[:i]</code>	Rango a partir de <code>i</code> , sin incluir el último elemento
<code>S[:]</code>	Selecciona todos los elementos
<code>S[i : j : k]</code>	De <code>i</code> a <code>j</code> de <code>a</code> <code>k</code>
<code>S[: -1]</code>	Lee <code>S</code> de derecha a izquierda
<code>S[i : j : -1]</code>	Desde <code>j</code> a <code>i</code> (orden inverso)
<code>len(S)</code>	Longitud
<code>str(33)</code>	Convierte a string '33'
<code>ord('s')</code>	Pasa a Unicode
<code>chr(115)</code>	Obtiene carácter a partir de Unicode
<code>"a %s ellos" % todos</code>	Da formato a expresión
	<code>s</code> = string, <code>c</code> = carácter, <code>d</code> =decimal, <code>i</code> =entero
	<code>f</code> = flotante decimal,
<code>"a {0} ellos" .format(todos)</code>	Da formato por posición en python 2.6, 2.7 y 3.x
<code>"a {valor} ellos" .format(valor = 1)</code>	Da formato por asignación
<code>S.find('pa')</code>	Búsqueda de <code>pa</code> dentro de <code>S</code> Retorna ubicación de primer carácter
<code>S.rstrip()</code>	Remueve espacio
<code>S.replace('pa', 'xx')</code>	Reemplaza <code>pa</code> por <code>xx</code>
<code>S.split(' , ')</code>	Separar según delimitador indicado



<code>S.isdigit()</code>	Test de contenido
<code>S.lower()</code>	Convierte S a minúscula
<code>delim.join(S)</code>	Crea un string con los elementos de S separados por delim
<code>' '.join(S)</code>	Crea un string con los elementos de S separados por espacio en blanco
<code>for x in S: print(x)</code>	Iteración
<code>'p' in perro</code>	Se fija si existe la letra p en el string perro
<code>chars = list('manzana')</code>	<code>chars = ['m', 'a', 'n', 'z', 'a', 'n', 'a']</code>
<code>chars.append('r')</code>	<code>chars = ['m', 'a', 'n', 'z', 'a', 'n', 'a', 'r']</code>
<code>' '.join(chars)</code>	<code>'manzanar'</code>
<code>[c * 2 for c in S]</code>	
<code>map(ord, S)</code>	
<code>re.match('sp(.*?)am', line)</code>	

3.2. Inmutabilidad de los strings.

El objeto "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 substring '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
<code>L = []</code>	Definición de lista vacía
<code>L = [False]*10</code>	Lista con 10 elementos FALSE
<code>L = [123, 'abc', {}]</code>	Lista con tres elementos
<code>L = ['Juan', 38, ['pintor', '5000']]</code>	Listas de más de una dimensión
<code>L = list('pera')</code>	Genera una lista con las letras de pera <code>L = ['p', 'e', 'r', 'a']</code>
<code>L=[i:j]</code>	Longitud
<code>len(L)</code>	Cantidad de elementos de la lista
<code>L1 + L2</code>	Suma dos lista
<code>L * 2</code>	Genera una lista que posee los elementos de L repetidos dos veces
<code>for x in L: print(x)</code>	Recorre e imprime cada elementos de L
<code>L.append(3)</code>	Le agrega a la lista el número 3
<code>L.extend([1, 2])</code>	Le agrega a L más de un elemento de lista
<code>del L[i]</code>	Elemento
<code>del L[i:j]</code>	Rango
<code>L[i] = 3</code>	Asignar valor a componente i
<code>L[i:j] = [4, 5]</code>	Asigno valor a componente i:j
<code>L = [x**2 for x in range(5)]</code>	<code>[0, 1, 4, 9, 16]</code>
<code>print(L[0])</code>	Imprimir el primer elemento desde la izquierda
<code>print(L[-1])</code>	Imprimir el primer elemento desde la derecha
<code>L=[[1,2],[3,4]]</code>	Matriz de 2*2
<code>print(L[0].split()[-2])</code>	Juan

Ejemplos de reasignación en listas

Supongamos que tenemos una lista F1 que apunta al objeto [0, 1, 2] si referenciamos a la lista desde F2 y modificamos el objeto al cual hace referencia F1, por ejemplo:

```
F1 = 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:

```
F1[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
# #####
F1 = [0, 1, 2]
F2 = F1                      # F1 y F2 son el mismo objeto
print(F2)
# #####
# Renombramos primer lista
# #####
F1[0]=4                      # F1 y F2 son el mismo objeto
print(F2)
# #####
# Renombramos primer lista    # F1 y F2 son distintos objetos
# #####
F1 = [2, 1, 2]
print(F2)
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
# #####
F1 = [0, 1, 2]
F2 = copy.copy(F1)           # F1 y F2 son distintos objetos
print(F2)

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:

```
J = 'Juan'           #J es un string
L = list(J)           #Transformo J en una lista
print(L)              #imprimo la lista
L[0] = 'P'            #LA LISTA PUEDE SER MODIFICADA
palabra = ''.join(L)  # 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
<code>D = { }</code>	Diccionario vacío
<code>D = { 'nombre': 'Juan', 'edad' : 39 }</code>	Dos ítems del diccionario
<code>E = { 'identificación': { 'nombre': 'Juan' } }</code>	Anidamiento
<code>D = dict('nombre': 'Juan', 'edad' : 39)</code>	Estructura alternativa
<code>D = dict([('nombre', 'Juan'),('edad', 39)])</code>	Pares clave/valor
<code>D = dict(zip(['clave1','clave2'],['valor1', 'valor2']))</code>	Crear lista con zip
<code>D['identificación']['nombre']</code>	Indexado por identificación y nombre
<code>'edad' in D</code>	Chequear si un campo se encuentra en el diccionario
<code>D.keys()</code>	Todas las claves
<code>D.values()</code>	todos los valores
<code>D.items()</code>	Tuplas de todas las (clave, valor)
<code>D.copy()</code>	copiar
<code>D.clear()</code>	remueve todos los ítems
<code>D.update(D2)</code>	Agrego D2 a D
<code>D.get(clave)</code>	Obtiene el valor de una clave
<code>D.pop(key)</code>	Borra un elemento por su clave y retorna el valor
<code>list(D.keys())</code>	Crea una lista que contiene las claves del diccionario
<code>list(D.values())</code>	Crea una lista que contiene los valores del diccionario
<code>list(D.items())</code>	Crea una lista que contiene las claves y los valores del diccionario
<code>len(D)</code>	Longitud



D[clave] = 42

Agrega una nueva clave con su valor

5.1. Crear diccionario.

Veamos a continuación las diferentes formas de crear un diccionario:

```
diccionario1_crear.py
# #####
# Crear diccionario
# #####
F = {'clave1': 'valor1', 'clave2': 'valor2', 'claven': 'valorn'}
print(F['clave1'])
# #####
# Crear por asignación
# #####
J = {}
J['clave1'] = 'valor1'
J['clave2'] = 'valor2'
print(F['clave1'])
# #####
# Crear mediante dict()
# #####
S = dict(clave1 = 'valor1', clave2 = 'valor2')
print(S['clave1'])
# #####
# Crear mediante dict() y zip
# #####
P = dict(zip(['clave1', 'clave2'], ['valor1', 'valor2']))
print(P['clave1'])
input()
```

El programa nos retorna:

```
valor1
valor1
valor1
valor1
```



5.2. Encadenamiento y agregar elementos

Los diccionarios pueden encontrarse encadenados, por lo que un diccionario puede contener otros diccionarios y los elementos de un diccionario pueden ser agregados mediante “append”. A continuación podemos verlo en funcionamiento con un ejemplo:

diccionarios2_anidar.py

```
# #####  
# Encadenamiento  
# #####  
F = {'nombre': {'primero': 'Juan', 'segundo': 'Marcelo'},  
     'trabajo': ['profesor', 'ingeniero']  
}  
print(F)  
print('nombre')  
print(F['nombre']['primero'])  
# #####  
# Agregar elementos  
# #####  
F['trabajo'].append('pintor')  
print(F)  
input()
```

El programa nos retorna:

```
{'trabajo': ['profesor', 'ingeniero'], 'nombre': {'primero': 'Juan', 'segundo': 'Marcelo'}}  
nombre  
Juan  
{'trabajo': ['profesor', 'ingeniero', 'pintor'], 'nombre': {'primero': 'Juan', 'segundo': 'Marcelo'}}
```

5.3. Pasar a lista.

En el caso de que decidamos pasar el diccionario creado a una lista, podríamos realizarlo mediante la utilización del método “list()”. Veámoslo en funcionamiento mediante el siguiente ejemplo:

diccionarios3_pasar_a_lista.py

```
# #####  
# Pasar a lista
```



```
# #####  
F = {'nombre': {'primero': 'Juan', 'segundo': 'Marcelo'},  
     'trabajo': ['profesor', 'ingeniero']  
}  
P = list(F.keys())  
print(P)  
S = list(F.values())  
print(S)  
input()
```

El programa nos retorna:

```
['trabajo', 'nombre']  
[['profesor', 'ingeniero'], {'primero': 'Juan', 'segundo':  
                              'Marcelo'}]
```

6. 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

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



- 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().



TRABAJOS A PRESENTAR EN EL FORO

Ejercicio 1

Cree una lista de frutas de 7 elementos, y realice un programa que muestre el tercer elemento de la lista en pantalla.

Ejercicio 2

Para concatenar texto se utiliza el signo +

Cree una lista de frutas de 2 elementos, y realice un programa con tkinter que muestre una oración conteniendo los dos elementos de la lista concatenándolos con texto para formar una oración con sentido.

Ejercicio 3

Cree un diccionario con las claves:

- identificador
- nombre
- apellido
- telefono

Nota: al utilizar claves no utilice acentos u otros caracteres en español, por ejemplo no ponga “teléfono” sino “telefono”.

Realice un programa que a partir del diccionario creado retorne en una oración:

- 1) El número de elementos del diccionario
- 2) Las claves del diccionario

Pregunta:

¿Cree que para guardar y recuperar información es mejor un diccionario o una lista?

Justifique su respuesta.

Ejercicio 4

Para tomar texto por consola se utiliza el comando: **input()**

Tome un nombre por consola y presente en la pantalla de la consola la oración:

Hola **nombre** bienvenido!

Ejercicio 5

Para pasar un valor de string a entero se utiliza el comando **int()**, así:

int("2")

Tome dos valores por consola, y guarde en una lista:

[primer_valor, segundo_valor, la_suma_de_los_valores]

Muestre en pantalla la lista.

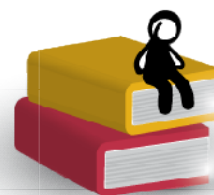
Ejercicio 6

Repita el ejercicio 5 pero en lugar de una lista utilice un diccionario.

Ejercicio 7 OPCIONAL

Intente realizar los ejercicios 1, 2 y 3 con tkinter.

Nota: No hemos entrado aún a analizar esta interfaz gráfica por lo que NO se solicita que se busque información extra en internet. Solo se está pidiendo presentar los datos en pantalla como se realizó en el ejercicio presentado en la unidad 1.



Bibliografía utilizada y sugerida

Libros

Programming Python 5th Edition – Mark Lutz – O'Reilly 2013

Programming Python 4th Edition – Mark Lutz – O'Reilly 2011

Manual online

<https://docs.python.org/3.7/tutorial/>

<https://docs.python.org/3.7/library/index.html>



Lo que vimos

En esta unidad comenzamos a analizar los tipos de datos que podemos utilizar en python.



Lo que viene:

En la siguiente unidad seguiremos trabajando y aprendiendo sobre los tipos de datos que podemos utilizar.