UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

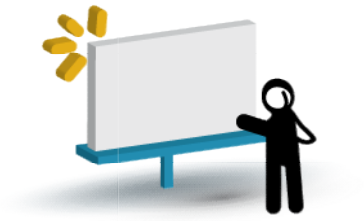
UNIDAD DIDÁCTICA III

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 III – Tipos de datos II.



Presentación:

En el transcurso de esta unidad seguiremos avanzando en el conocimiento de las bases del lenguaje, en particular en el aprendizaje de los tipos de datos que podemos utilizar y en el uso de print() y date().

Objetivos:

Que los participantes:

- Aprendan sobre los tipos de datos: Tuplas y Set.
- Implementen herramientas de formateo de salida de datos y fechas.
- Avancen en el conocimiento de la GUI Tkinter.

Bloques temáticos:

1.- Tupla.

2.- Set.

3.- Formato de salida de datos.

4.- Date.

5.- GUI.

Consignas para el aprendizaje colaborativo

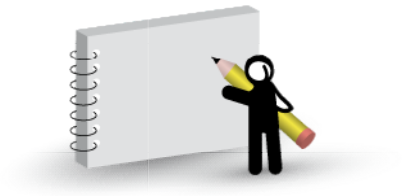
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- 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. Tupla.

Las tuplas son secuencias como las listas pero son INMUTABLES como los strings. Se utilizan para representar colecciones fijas de ítems. Por ejemplo los componentes de un calendario específico.

Se crean con o sin paréntesis, aunque en ocasiones en declaraciones complejas es necesario incluir los paréntesis.

```
tupla1 = 1, 2, 3, 4, 5, 3
print(type(tupla1))
tupla2 = (1, 2, 3, 4, 5, 3)
print(type(tupla2))
```

Retorna:

```
<class 'tuple'>

<class 'tuple'>
```

Las tuplas pueden ser anidadas, y podemos ver que la salida es retornada siempre entre paréntesis.

```
tupla3 = tupla1, tupla2
print(type(tupla3))
print(tupla3)
```

Retorna:

```
<class 'tuple'>

((1, 2, 3, 4, 5, 3), (1, 2, 3, 4, 5, 3))
```

Para determinar un elemento de la tupla lo hacemos de forma análoga a como lo realizamos con las listas, y de la misma forma que con el resto de los tipos de datos contamos con métodos que podemos utilizar mediante notación de punto.

```
tupla4 = tupla3 + (6, 7)
print(tupla4)
# ########################################
# Determinar un elemento
# ########################################
print(tupla4[0])
# ########################################
# Posición de un elemento
# ########################################
print(tupla4.index(6))
# ########################################
# n° de veces que aparece un elemento
# ########################################
print(tupla4.count(7))
```

Retorna:

```
((1, 2, 3, 4, 5, 3), (1, 2, 3, 4, 5, 3), 6, 7)

(1, 2, 3, 4, 5, 3)

2

1
```

Un problema especial es la construcción de tuplas que contienen 0 o 1 elementos: la sintaxis tiene algunas peculiaridades adicionales para acomodarlos. Las tuplas vacías se construyen con un par de paréntesis vacíos; una tupla con un elemento se construye siguiendo un valor con una coma (no es suficiente para encerrar un solo valor entre paréntesis).

```
tupla5 = ()
tupla6 = 'Manzana',
print(len(tupla5))
print(len(tupla6))
print(tupla6)
```

Retorna:

```
0

1
```

```
('Manzana',)
```

2. SET

Es una colección sin orden de objetos únicos e INMUTABLES que soportan operaciones que se corresponden con la teoría matemática de set. Por definición un ítem aparece una sola vez en el set, no importa cuántas veces se agrega. Es como trabajar con los conjuntos que aprendimos en el colegio y de los cuales realizamos operaciones de unión, intersección y diferencia en los “Diagramas de Venn”.

La aplicación de los sets es en casos numéricos y trabajos con bases de datos.

Los sets son como las claves de un diccionario presentadas en forma desordenada. Sus elementos se escriben entre llaves. Vamos ahora a ver un ejemplo:

sets/set1_definicion.py

```
# ########################################
# Crear set
# ########################################
set1 = {1, 2, 3, 4, 5, 3}
print(set1)
print(len(set1))
```

Resultado:

```
{1, 2, 3, 4, 5}
5
```

Nota: Veamos que aunque el set anterior posee 6 elementos, al obtener su longitud con len() el resultado da 5 dado que el set no cuenta los elementos repetidos (en este caso el 3)

Un ejemplo un poco más complejo nos permite ver cómo trabajar algunas operaciones básicas con los sets.

sets/set2_operaciones.py

```
# ########################################
# Crear set
# ########################################
set1 = {1, 2, 3, 4, 5, 3}
set2 = {10, 2, 13, 4, 15, 3, 1}
set3 = {10, 2, 13, 4, 15, 3}
# ########################################
# Crear Intersección
```

```
# ########################################
print(set2 & set2)
# ########################################
# Crear Unión
# ########################################
print(set1 | set2)
# ########################################
# Crear Diferencia
# ########################################
print(set1 - set2)
# ########################################
# Crear Superset
# set2 es un set que contiene todos los
# elementos de set3 y más
# ########################################
print(set2 > set3)
```

Las salidas son respectivamente:

```
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 10, 13, 15}
{5}
True
```

Set vacío.

Un set vacío debe inicializarse mediante set() ya que {} define un diccionario

Usos de los sets.

Dado que los ítems son guardados solo una vez, pueden usarse para filtrar duplicados, y ya que son desordenados, deberían ordenarse en el proceso. Por ejemplo podría convertirse una lista a set y luego regresar a una lista.

sets/set3_ejemplo1_remover_duplicados.py

```
# ########################################
# Remover duplicados
# ########################################
lista = [1, 2, 1, 3, 2, 4, 5]
mi_set = set(lista)
```

```
print(lista)
print(mi_set)
lista2 = list(mi_set)  # remueve duplicados
print(lista2)
```

La salida nos da:

```
[1, 2, 1, 3, 2, 4, 5]

{1, 2, 3, 4, 5}

[1, 2, 3, 4, 5]
```

Aplicación

Este tipo de datos nos podría servir por ejemplo para analizar las listas de membresía de un club y realizar estadísticas sobre los deportes o actividades realizadas por los miembros. Supongamos que tenemos las siguientes listas:

```
socios = [“Juan”, “Pedro”, “Susana”, “Anna”, “Sofía”, “Pablo”]
```

Y que en el club se realiza natación y ajedrez:

```
ajedrez = ["Pedro", "Susana", "Anna", "Sofía", "Pablo"]
natacion = ["Juan", "Pedro", "Susana"]

resultado=set(ajedrez).intersection(set(natacion))
print(resultado)
print(type(resultado))
```

Retorna:

```
{'Susana', 'Pedro'}

<class 'set'>
```

También podríamos utilizar los métodos “unión()” o “difference()”, para unir o hallar diferencias de conjuntos.

3. Formato de salida de datos

Hasta ahora hemos utilizado el método print() para imprimir algunos resultados en pantalla, este método posee algunas otras variantes, como es el caso de imprimir varios resultados juntos separados por espacios u otros tipos de caracteres:

```
x = 1
y = 2
z = ['juan']
print(x, y, z, sep=' ')
print(x, y, z, sep=' , ')
```

Retorna:

```
1 2 ['juan']

1 , 2 , ['juan']
```

O agregar alguna instrucción al final de cada línea, como una descripción, tabulación o salto de línea.

```
print(x, y, z, sep=' , ', end='!\n')
```

Retorna:

```
1 , 2 , ['juan']!
```

Secuencia de datos

En ocasiones es útil trabajar con secuencia de datos, supongamos que tenemos la lista

```
seq = [1, 2, 3, 4]
```

Y que asociamos una posición a una referencia de la siguiente manera:

```
a, b, c, d = seq
```

De esta forma al imprimir

```
print(a, b, c, d)
```

Nos retorna:

```
1 2 3 4
```

En el ejemplo anterior también es posible asociar a un grupo de valores una lista , por ejemplo podemos decir que el primer valor está asociado a la letra “a” y el resto a una lista llamada “b” anteponiendo un asterisco a la letra b de la siguiente forma:

```
a, *b = seq
print(a,b, type(b))
```

El resultado que nos retorna es:

```
1 [2, 3, 4] <class 'list'>
```

En el caso de que en lugar de una lista estemos trabajando con un string, b crearía una lista de caracteres.

```
a, *b = 'Manzana'
print(a, b, type(b))
```

Retorna.

```
M ['a', 'n', 'z', 'a', 'n', 'a'] <class 'list'>
```

También contamos con algunas variantes muy útiles, como los formatos literales, veamos cuales son y cómo se utilizan:

Formato literal (f-strings)

Esta es una forma muy útil de darle formato a los datos, debemos comenzar una cadena de texto con “f” o “F” antes de las comillas de apertura o las comillas triples. Dentro de esta cadena podemos escribir una expresión de Python entre los caracteres de apertura y cierre de llaves.

```
nombre = 'Pedro'
edad = 40
print(f'La edad de {nombre} es de {edad} años')
```

Retorna:

```
La edad de Pedro es de 40 años
```

También podríamos utilizar este formato para realizar columnas de dados, agregando luego de lo que estamos imprimiendo, dos puntos seguidos de la mínima cantidad de caracteres que queremos imprimir.

```
nombre = 'Pedro'
edad = 40
print(f'{"fila":30}==>{nombre:10}{edad}')
```

```
fila                          ==>Pedro     40
```

O especificar la cantidad de decimales luego de la coma de un número.

```
valor = 3.141659
print(f'El valor redondeado a 3 dígitos luego de la coma queda: {valor:.3f}.')
```

Retorna:

```
El valor redondeado a 3 dígitos luego de la coma queda: 3.142.
```

O especificar la cantidad de decimales luego de la coma de un número.

Formato con format()

Dar formato con “format()” es bastante similar, también utilizamos las llaves para indicar donde se coloca la variable dentro del string pero podemos hacer referencia por posición utilizando números, veamos un ejemplo:

```
votos = 42572654
voto_en_blanco = 43132495
porcentaje = (votos / (votos + voto_en_blanco)) *100
print('Los votos a favor son: {0} con un porcentaje de: {1:.2f}'.format(votos, porcentaje))
```

Retorna:

```
Los votos a favor son: 42572654 con un porcentaje de: 49.67
```

Notemos que las posiciones se comienzan a numerar desde cero y que en particular el segundo parámetro posee un uno (el cual representa la posición dentro de format()) seguido de dos puntos y a continuación la cantidad de dígitos después de la coma (.2f)

pprint

Existe un módulo que viene en las distribuciones de python y que podemos importar para aumentar la legibilidad llamado pprint, analicemos un ejemplo utilizando diccionarios en donde imprimimos utilizando print() y pprint()

```
import pprint
```

```
juan = {'identificacion': {'nombre': 'Juan', 'apellido':
'Garcia'},
'edad': 24,
'sueldo': 5000,
'profesión': 'Pintor'}
susana = {'identificacion': {'nombre': 'Susana', 'apellido':
'Gomez'},
'edad':25,
'sueldo': 6000,
'profesión': 'Empleada'}
db= {}
db['juan'] = juan
db['susana'] = susana
print(db)
print('------------------------------')

pprint.pprint(db)
```

Retorna:

```
{'juan': {'identificacion': {'nombre': 'Juan', 'apellido': 'Garcia'}, 'edad': 24, 'sueldo': 5000,
'profesión': 'Pintor'}, 'susana': {'identificacion': {'nombre': 'Susana', 'apellido': 'Gomez'},
'edad': 25, 'sueldo': 6000, 'profesión': 'Empleada'}}

------------------------------

{'juan': {'edad': 24,

          'identificacion': {'apellido': 'Garcia', 'nombre': 'Juan'},

          'profesión': 'Pintor',

          'sueldo': 5000},

'susana': {'edad': 25,

            'identificacion': {'apellido': 'Gomez', 'nombre': 'Susana'},

            'profesión': 'Empleada',

            'sueldo': 6000}}
```

4. Date

Trabajar con fechas es algo muy importante y que realizamos muy a menudo, existen dos librerías muy útiles:

```
import datetime
import calendar
```

Con datetime podemos obtener datos de la fecha actual, desde el año a microsegundos

```
import datetime
import calendar

datetime.datetime.now()
print(datetime.datetime.utcnow())
print(datetime.datetime.now())
print(datetime.datetime.now().day)
print(datetime.datetime.now().month)
print(datetime.datetime.now().minute)
print(datetime.datetime.now().second)
print(datetime.datetime.now().microsecond)
```

Contamos con una serie de abreviaciones que nos permiten especificar claramente el formato de fecha e incluso realizar operaciones entre fechas:

```
print(datetime.datetime.today().strftime("%H:%M:%S--%d/%m/%y"))
"""
%a - Nombre del día de la semana
%A - Nombre del día completo
%b - Nombre abreviado del mes
%B - Nombre completo del mes
%c - Fecha y hora actual
%d - Día del mes
%H - Hora (formato 24 horas)
%I - Hora (formato 12 horas)
%j - Día del año
%m - Mes en número
%M- Minutos
%p - Equivalente de AM o PM
%S - Segundos
```

```
%U - Semana del año (domingo como primer día de la semana)
%w - Día de la semana
%W - Semana del año (lunes como primer día de la semana)
%x - Fecha actual
%X - Hora actual
%y - Número de año (14)
%Y - Número de año entero (2014)
%Z - Zona horaria
"""
#Diferencia de fechas 1
actual = datetime.datetime.now()
anterior = datetime.datetime(1975, 9, 15, 0, 0, 0)
print(actual-anterior)

#Diferencia de fechas 2
hoy = datetime.date.today()
hace5 = datetime.timedelta(days=5)
print(hoy)
print(hace5)
print(hoy - hace5) #Resto 5 días
```

5. GUI

Label

Avancemos un poco en la utilización de la interfaz gráfica que comenzamos a ver en la unidad 1. Tkinter cuenta con varias estructuras preestablecidas, y existe una en particular llamada “Frame” que nos permite organizar el diseño de nuestra aplicación, y es fundamental a la hora de maquetar, ya que cumple varias funciones:

1.- Nos permite agrupar aplicaciones de tkinter para crear diseños complejos.

2.- Los podemos utilizar como separadores de secciones si les damos dimensiones y un contenido vacío.

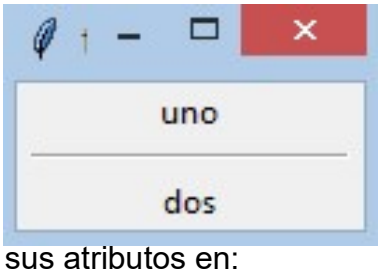
3.- Los podemos utilizar como basé en la creación de aplicaciones.

Su implementación es muy simple, en el siguiente ejemplo lo utilizamos para separar dos textos, de forma similar a como lo haríamos en un documento html mediante el uso de la etiqueta <hr/>. Notar que agregamos explícitamente el “Frame” en la ventana pasando la ventana como primer parámetro. Esta es la forma de indicar que un componente cualquiera se encuentra dentro de otro.

```
from tkinter import *

master = Tk()
Label(text="uno").pack()
separador = Frame(master, height=2, bd=1, relief=SUNKEN)
separador.pack(fill=X, padx=5, pady=5)
Label(text="dos").pack()
mainloop()
```

Su representación gráfica queda:



Podemos encontrar una referencia completa a sus atributos en:

http://effbot.org/tkinterbook/frame.htm

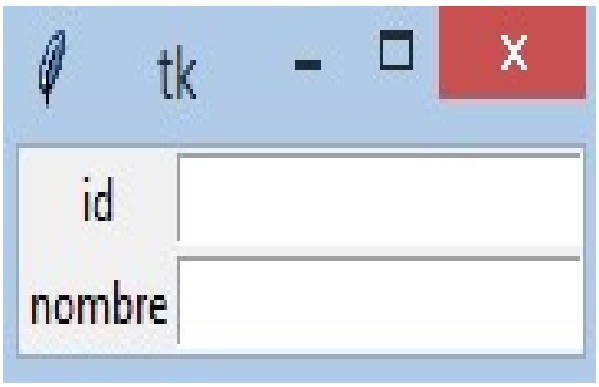
En el transcurso de las unidades y con la introducción de los bucles, estructuras de control y funciones iremos viendo cómo desarrollar aplicaciones más interesantes.

Grillas

El generador de grillas “Grid”, permite crear aplicaciones que se ubiquen en una tabla de dos dimensiones. El elemento padre es particionado en un determinado número de filas y columnas, y cada "celda" de la tabla resultante puede contener una nueva aplicación.

Es muy útil en la generación de formularios, por ejemplo para lograr crear un formulario con dos campos que posean dos entradas, lo podemos hacer de la siguiente forma:

```
from tkinter import *
root = Tk()
Label(root, text="id").grid(row=0, column=0)
Label(root, text="nombre").grid(row=1, column=0)
e1 = Entry(root)
e2 = Entry(root)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
mainloop()
```



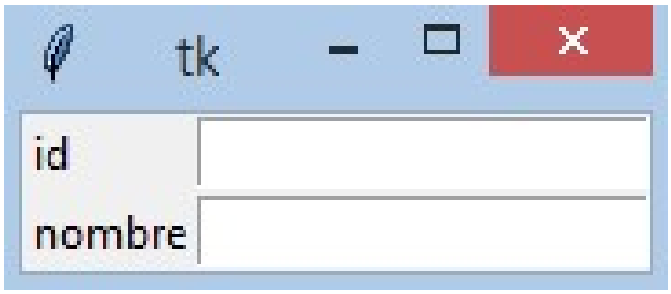
Uso de sticky

Dentro de una grilla, las aplicaciones se ubican centradas (esto se nota en el ejemplo anterior en el caso del Label que contiene el id), para modificar este comportamiento podemos utilizar el atributo "**sticky**", el cual toma como parámetros las opciones: N, S, W y E.

Así, si queremos que los nombres de los campos en el ejemplo anterior se posicionan a la izquierda, podemos modificar el código de la siguiente forma:

```
from tkinter import *
root = Tk()
Label(root, text="id").grid(row=0, column=0, sticky=W)
Label(root, text="nombre").grid(row=1, column=0, sticky=W)
e1 = Entry(root)
e2 = Entry(root)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
mainloop()
```

Representación gráfica:



Podemos encontrar una referencia completa a sus atributos en:

http://effbot.org/tkinterbook/grid.htm

Botones y sus parámetros.

Cuando queremos agregar un botón, contamos con la estructura Button(), veamos un detalle de los parámetros que puede recibir.

Nomenclatura: Button(master=None, **options)

Función callback

command=nombreFunción

En este caso creamos un botón asociado al widget master con texto "text" y que ejecuta la función "callback", una función como veremos en la siguiente unidad es una rutina definida con un nombre al cual se le antepone la palabra reservada “def” y finaliza con dos puntos (:), y lo que hace la rutina se pone debajo pero agregando una tabulación o serie de espacios para indicar que pertenece a la función.

Nota: No nos preocupemos a esta altura de su estructura, simplemente veamos una aplicación para irnos acostumbrando a su existencia.

```
from tkinter import *

master = Tk()
def callback():
    print("click!")
b = Button(master, text="OK", command=callback)
b.pack()
mainloop()
```

Deshabilitarlo

Para en una construcción de prueba, poder deshabilitar el botón, le agrego state=DISABLED

```
b = Button(master, text="Help", state=DISABLED)
```

Padding

El padding es un espaciado interno al botón, que permite delimitar el área de escritura dentro del mismo.

```
from tkinter import *
master = Tk()
def callback():
    print("click!")
b = Button(master, text="OK", command=callback, padx=132, pady=132)
b.pack()
mainloop()
```

Alto y ancho

El alto y ancho de un botón de texto está dado en unidades de texto, mientras que si es una imagen, se trabaja en px.

```
from tkinter import *

master = Tk()

b = Button(master, text="Sure!", anchor=W, justify=LEFT, padx=22, height=3, width=12)
b.pack(fill=BOTH, expand=1)
mainloop()
```

Imagen

La imagen si no implementamos otra librería adicional debe estar en formato .gif

```
from tkinter import *
master = Tk()

photo=PhotoImage(file="download.gif")
c = Button(master, text="Sure!", anchor=W, justify=LEFT, image=photo )
c.pack(fill=BOTH, expand=1)
mainloop()
```

Color.

Fondo: background="black" ó bg="black"

Letra: foreground="red" ó fg="red"

Fondo activo: activebackground="green"

Letra activa: activeforeground="yellow",

Letra deshabilitado: disabledforeground="blue"

```
from tkinter import *

master = Tk()
def callback():
    print("click!")
b = Button(master, text="OK", command=callback, padx=132, pady=132,
activebackground="green", activeforeground="yellow",
background="black", foreground="red"
)
b.pack()
a = Button(master, text="OK", command=callback, padx=132, pady=132,
state=DISABLED, background="black", disabledforeground="blue"
)
a.pack()
mainloop()
```

Posición de texto/imagen y botón

Posición de texto e imagen

anchor=SW

Opciones: N, NE, E, SE, S, SW, W, NW, o CENTER. Por default es CENTER. (anchor/Anchor)

Posición de botón

Esta opción va dentro de pack()

side=LEFT

Opciones: LEFT, TOP, RIGHT, BOTTOM

```
from tkinter import *

master = Tk()
master.geometry("300x300")
def callback():
    print("click!")
b = Button(master, text="OK", command=callback,
    activebackground="green", activeforeground="yellow",
    background="black", foreground="red", height=7, width=12, anchor=SW)
b.pack(side=LEFT)
mainloop()
```

Tipo de texto

font=('tipo', tamaño, 'peso')

```
from tkinter import *

master = Tk()

b = Button(master, text="Ok!", anchor=W, justify=LEFT, padx=22,
height=3, width=12, font=('courier', 22, 'bold'))
b.pack(fill=BOTH, expand=1)
mainloop()
```

El conjunto completo de parámetros se puede obtener de la siguiente referencia: ***http://effbot.org/tkinterbook/button.htm***

Pantallas complejas

La creación de pantallas complejas, requieren de un diseño previo a sentarse a escribir alguna línea de código, cuestiones como:

- Diseño gráfico
- Navegabilidad
- Usabilidad
- Reutilización de código
- Tipo de dato a almacenar
- Tipo de base de datos y diseño de la misma
- Diseño de la página web de descarga.
- Etc…

Hacen que el diseño de una aplicación vaya más allá de sí funcionalmente es correcta o no, una excelente aplicación que abarque todas las áreas de incumbencia y sea rápida y robusta, puede quedar por el camino si los programadores no han pensado en un buen diseño gráfico y una buena campaña de marketing. Pensemos por un momento en un editor de texto, los hay de todo tipo, dado que vamos a trabajar en algún momento con páginas del tipo html tratemos de elegir un editor de texto para trabajar con este tipo de formato.

Supongamos que no tenemos ninguna idea y que por lo tanto hacemos una búsqueda en google para ver opiniones de otras personas (recuerden que en principio solo queremos trabajar con formato .html). Cuando ingresamos en los foros a buscar opiniones, nos encontramos con personas que al utilizar formatos .html también trabajan con javascript para agregar funcionalidad del lado del usuario, con hojas de estilos para darle estilos a nuestra página, con programas como ruby para poder programar en hojas de estilos mediante la aplicación Sass, con AJAX para introducir código en forma asincrónica, etc... Por lo que encontrar una opinión en principio no resulta tan fácil como creíamos ya que nos encontramos con muchos términos nuevos y con un volumen muy grande de información que no podemos procesar. En vista de esto quizás nuestro siguiente paso sea hacernos un listado de los nombres de programas que se han citado en los artículos que hemos estado leyendo y buscar en las páginas oficiales de cada aplicación para ver si nos dan una mano a decidirnos por alguna. Supongamos que nuestro listado es el siguiente:

- VSC
- Pycharm
- Jupyter Lab
- Notepadd++
- Sublime2
- NetBeans
- Brackets
- Eclipse
- Dreamweaver
- Aptana Studio
- Bluegriffon

Es muy probable que luego de la búsqueda nos decidamos por uno o por otro por factores como si es gratis o no, o por los colores y estilos que vienen por defecto en la zona de escritura, aun cuando todos ellos tengan la posibilidad de configurar los mismos parámetros.

Hagan la prueba e intenten decidir por uno de los editores del listado anterior.

Como podemos ver realizar una aplicación requiere más esfuerzo que solo saber programar y es necesario interactuar con personas de otras disciplinas y potenciales usuarios para poder crear una aplicación exitosa.

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

http://effbot.org/tkinterbook/frame.htm

http://effbot.org/tkinterbook/grid.htm

Lo que vimos

En esta unidad avanzamos en el conocimiento de los tipos de datos que podemos utilizar en python y comenzamos a ver herramientas útiles para representar las salidas de datos y fechas.

Lo que viene:

En la siguiente unidad seguiremos comenzaremos a trabajar con funciones, estructuras de control y bucles.