Investigación en Inteligencia Artificial

Herramientas para la investigación en inteligencia artificial: introducción al lenguaje de programación Python

Índice

Esquema

Tipos de datos

- Constantes
- Variables
- Cadenas de texto
- Listas
- Conjuntos
- Tuplas
- Diccionarios
- Clases y objetos

Manipulando datos

- Condicionales
- Bucles
- Funciones

Visualización

- Líneas
- Histogramas
- Función de densidad
- Gráficos de tarta
- Scatter Plots

Ideas clave

1.1. ¿Cómo estudiar este tema?

Para estudiar este tema lee las **Ideas clave** disponibles a continuación. Todo lenguaje de programación se aprende practicando. Por tanto, **ejecuta por ti mismo todo el código que aparece en el presente tema y trabájalo**. Cambia, añade, quita… lo que te parezca oportuno y experimenta.

Python se convertirá en nuestro lenguaje de programación de referencia durante todo el programa, por lo que su manejo de forma autónoma se hace imprescindible. En este tema pretendemos conseguir los siguientes objetivos:

- Conocer las peculiaridades del lenguaje de programación Python.
- Comprender los principales tipos de datos en Python.
- Ser capaz de diseñar programas de complejidad media en Python.
- Se capaz de realizar un gráfico sencillo en Python.

1.2. Características generales de Python y entornos de desarrollo

Python es un lenguaje de programación de propósito general que figura entre los más populares a la hora de desarrollar prototipos basados en inteligencia artificial en general y aprendizaje automático en particular. Las bases del lenguaje comenzaron a fijarse en 1980, aunque hubo que esperar a 1989 para que se

iniciasen los trabajos de la primera implementación del lenguaje liderada por **Guido Van Rossum** desde Holanda. Su nombre es un guiño al célebre grupo cómico Monty Python.

Python es un lenguaje interpretado con una sintaxis sencilla que permite desarrollar y evaluar de forma iterativa y ágil modelos analíticos. Es, además, un producto de código abierto y de uso gratuito.

Es por eso por lo que cuenta con gran grado de aceptación entre la comunidad de expertos de la disciplina. Además, es uno de los lenguajes más demandados a la hora de ofertar un puesto de trabajo.

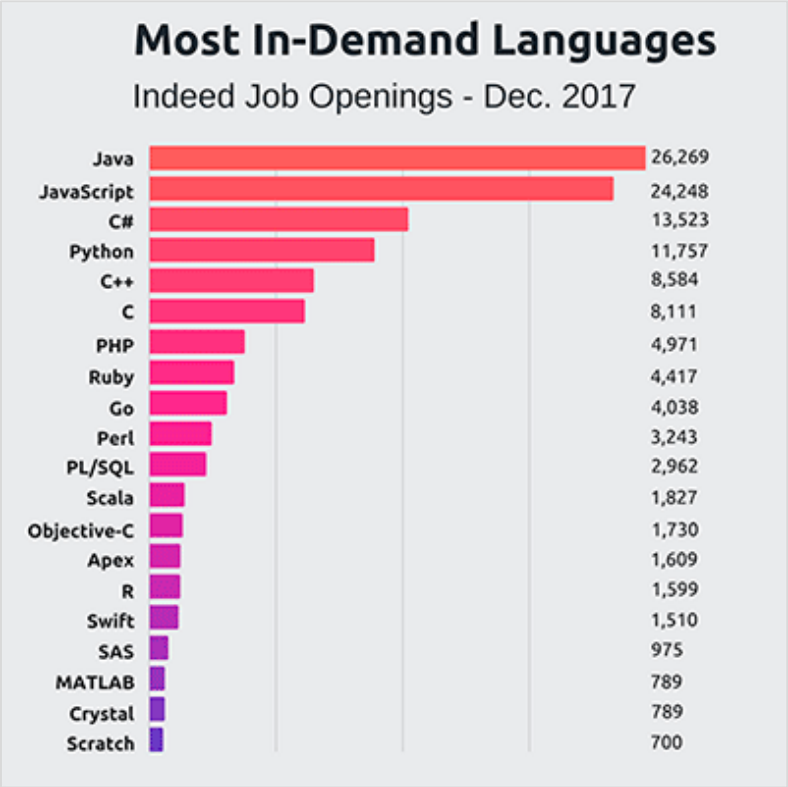

Figura 1. Lenguajes de programación más demandados en las búsquedas de empleo del sector informático según el portal Indeed.com. Fuente: https://stackify.com/popular-programming-languages-2018/

Dentro del ámbito de la inteligencia artificial, aprendizaje automático, etc., otros lenguajes usados son R, SAS y Matlab…

En los recursos recomendados tienes información para refrescar los conocimientos básicos sobre variables, funciones, bucles, instrucciones condicionales, etc.

Instalación de Phyton

Un lenguaje de programación como Python se compone de los módulos principales del lenguaje más un conjunto de librerías adicionales que permiten extender sus funcionalidades encapsulando utilidades de uso común. Existen principalmente dos formas de instalar Python. La primera implica instalar los paquetes core de Python y posteriormente incorporar las librerías adicionales que se precisen.

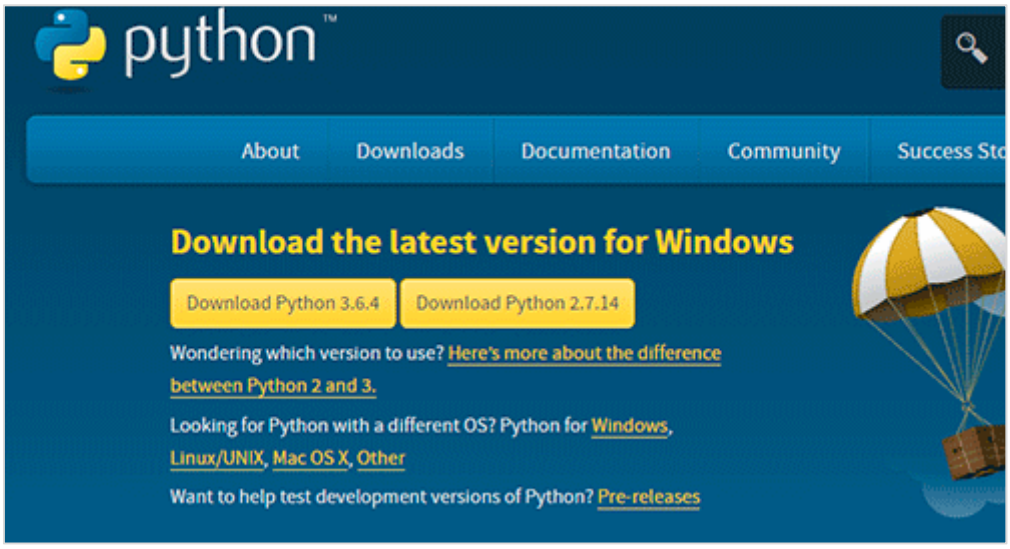

Figura 2. Página principal de descargas de Python. Fuente: https://www.python.org/downloads/

Como podemos comprobar, existen dos tipologías de versiones de Python. Python 3 permite generar código más eficiente y reusable, y será la versión empleada en esta asignatura.

Si nos decantamos por esta opción, para instalar el resto de paquetes y utilidades requeridas habrá que usar pip, un popular gestor de paquetes para Python.

Nosotros nos decantaremos por emplear el segundo método de instalación. Este método se basa en la solución proporcionada por Anaconda, un popular proveedor de utilidades basadas en Python. Para ello, en primer lugar, debemos descargar a nuestro equipo la versión 3.6.

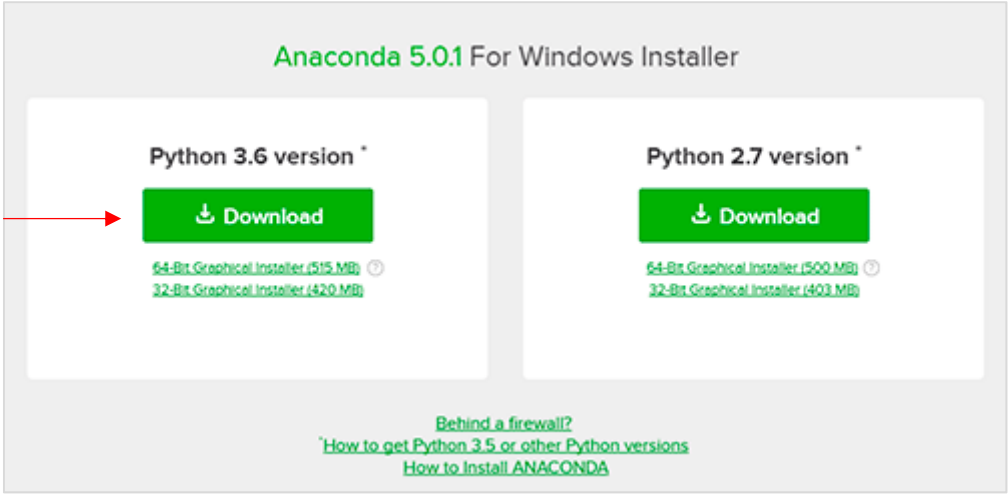

Figura 3. Página de descargas de Anaconda 5.0.1. Fuente: https://www.anaconda.com/download/

Cada usuario deberá elegir la versión más adecuada en función del sistema operativo que tenga instalado (Windows, Mac o Linux).

Para instalarlo simplemente se deben seguir los pasos indicados por el programa.

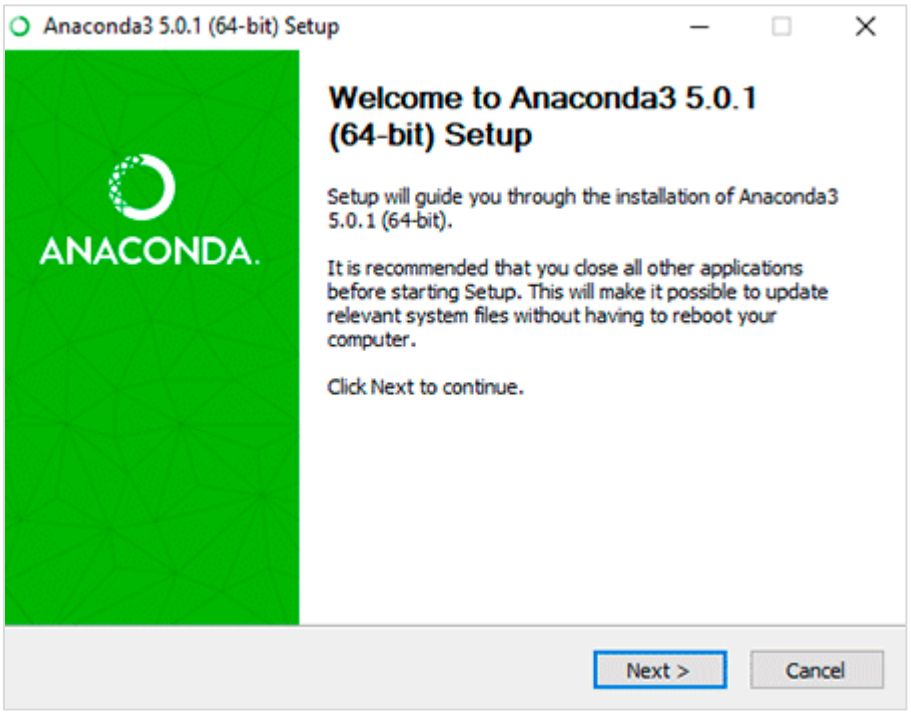

Figura 4. Herramienta de instalación de Anaconda con Python 3.6.

De cara al curso de introducción, se recomienda activar las dos opciones que se indican durante el proceso de instalación:

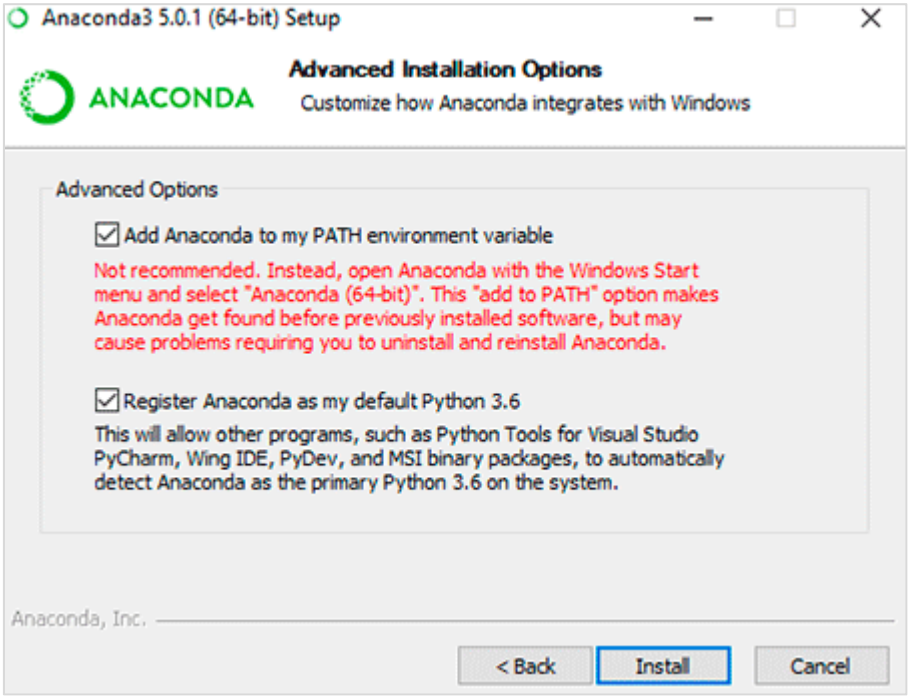

Figura 5. opciones de instalación en Anaconda.

La gran ventaja de Anaconda es que ya integra en su instalación la mayoría de los paquetes y utilidades necesarios para desarrollar aplicaciones que impliquen procesamiento de datos y desarrollo de modelos de aprendizaje automático.

Aun así y una vez instalado Anaconda, **podemos añadir nuevos paquetes** según el procedimiento que sigue. Por ejemplo, añadiremos a la instalación base de Anaconda el paquete NetworkX. Este paquete añade nuevas funcionalidades que permiten trabajar y analizar grafos.

Para ello debemos emplear la utilidad «conda» de la siguiente forma:

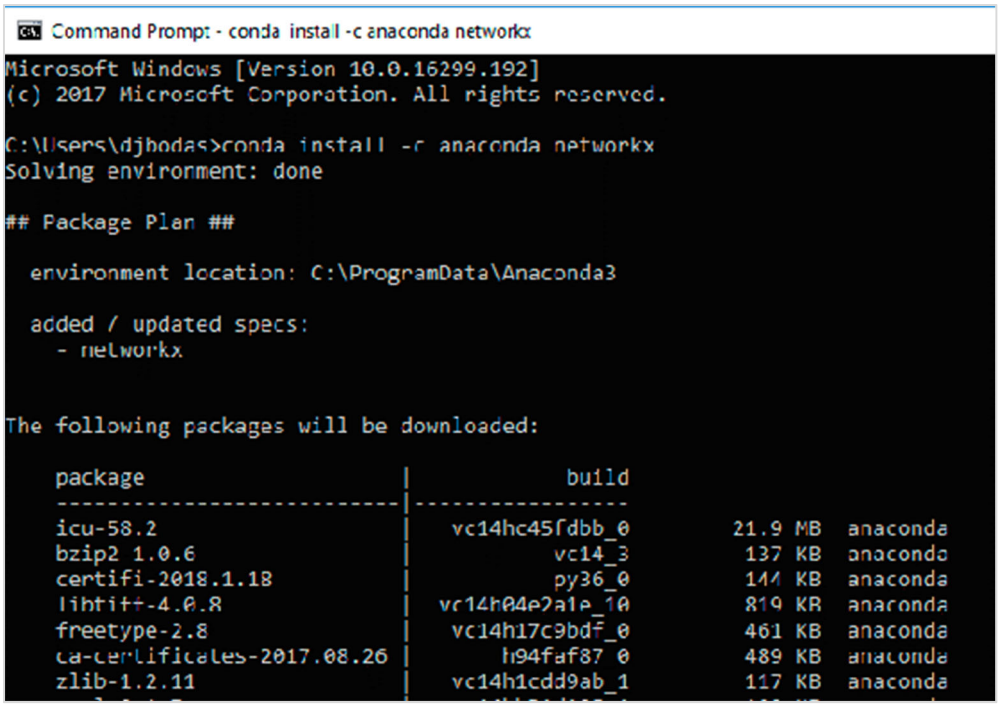

Figura 6. instalando un paquete adicional con «conda»

Una vez introducido el comando, debemos aceptar y descargar todas las dependencias.

Nota: la opción «-c» añade direcciones adicionales para localizar el paquete deseado.

Las opciones detalladas de instalación se pueden consultar en este enlace:

https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html

De cara a los usuarios de Mac, se recomienda encarecidamente la instalación de Homebrew el popular gestor de paquetes para Mac.

Descarga Homebrew desde este enlace:

https://brew.sh/index_es.html

Podría ser muy útil en algunas fases de la asignatura para instalar opciones adicionales.

Son varias las posibilidades a la hora de trabajar con Python, incluyendo mediante consola.

Entornos de desarrollo

Siendo prácticos, existen dos formas esenciales de trabajar, podemos emplear un entorno de desarrollo o un bloc de notas (comúnmente conocidos como notebooks) preparado para ejecutar código Python. Algunos IDE (por sus siglas en inglés: Integrated Development Environment) de Python conocidos son:

Pycharm (https://www.jetbrains.com/pycharm/download/): existe una versión profesional y una versión de uso gratuito para la comunidad.

Spyder: IDE básico que se proporciona junto a la instalación base de Anaconda.

Según nuestra instalación, lo primero que debemos hacer es lanzar Anaconda Navigator. Así tendremos acceso a la siguiente pantalla:

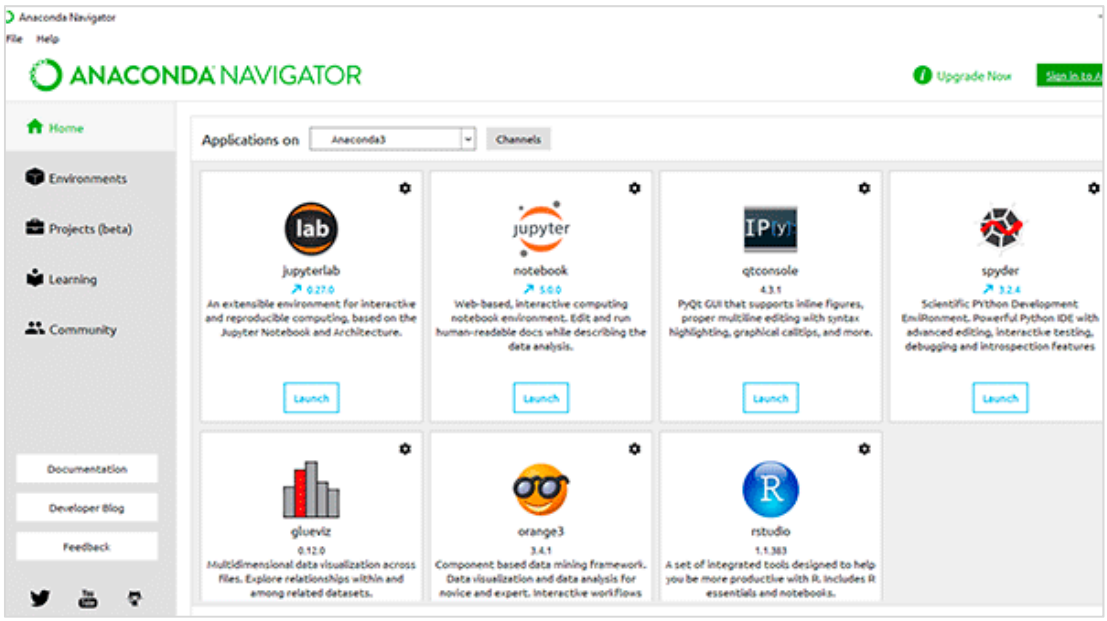

Figura 7. Anaconda Navigator.

En la parte superior derecha podemos ver en enlace a la herramienta Spyder. Sin embargo, nosotros trabajaremos con la popular versión del notebook Jupyter. Para ello debemos lanzar esta opción a través del botón correspondiente.

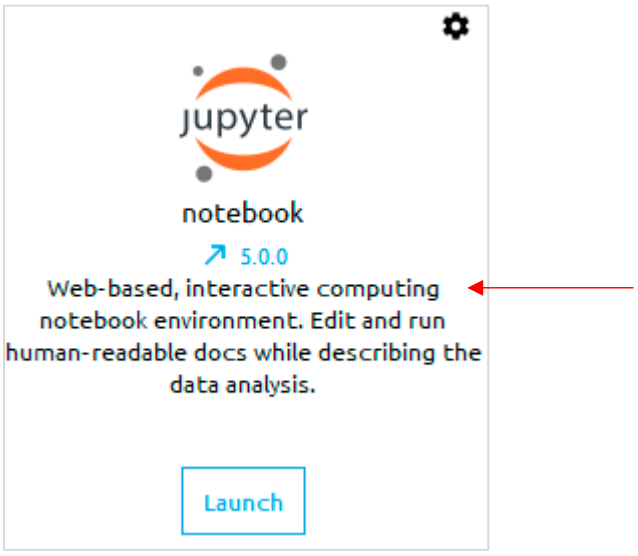

Figura 8. Lanzar notebook Jupyter.

Interacción básica con los notebooks Jupyter

Si todos los pasos se han ejecutado correctamente, se debería ver la siguiente pestaña en tu navegador:

Figura 9. Jupyter Notebook.

Nota: si la pestaña que se abriese en el navegador estuviese en blanco, probablemente se deba a un problema con el navegador. Por ejemplo, ciertas versiones de Internet Explorer pueden no ser recomendables aquí. Si esta situación se da, se recomienda copiar la dirección que aparece en la barra de direcciones del navegador e intentar con otro navegador distinto (Firefox, Chrome…)

A continuación, navegaremos por la estructura de directorios hasta donde queramos comenzar a guardar nuestros notebooks. Si es preciso, crearemos el directorio a través de la utilidad correspondiente.

Figura 10. Creando un directorio (Jupyter).

Debe observarse el peculiar funcionamiento en este caso, primero debemos crear el directorio y luego cambiar el nombre al valor deseado. A partir de este momento podemos pasar a crear el notebook de trabajo, que recibirá el nombre deseado pulsando en el título primario (Untitled, situado en la parte superior izquierda).

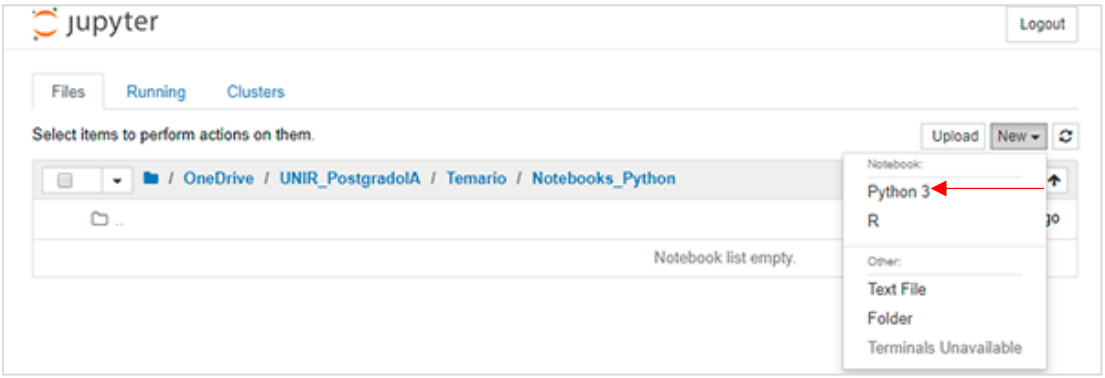

Figura 11. Crear un notebook (Jupyter).

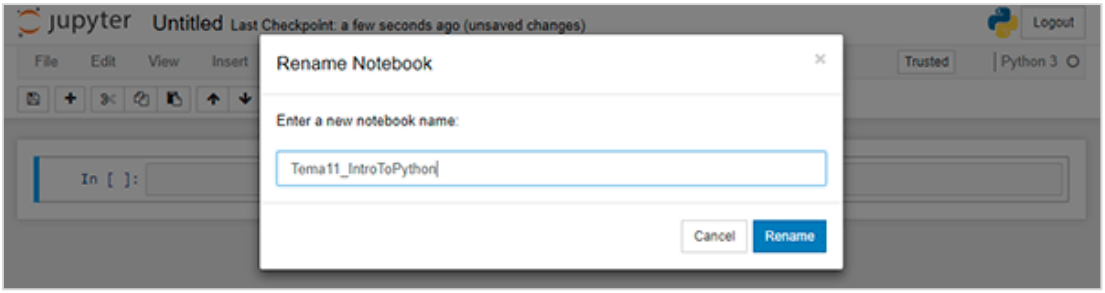

Figura 12. Renombrar el notebook (Jupyter).

Debemos observar que la pestaña desde la que hemos llamado a la creación del notebook permanece abierta y muestra enlaces interesantes. Desde la pestaña Running podemos gestionar todos los notebooks abiertos.

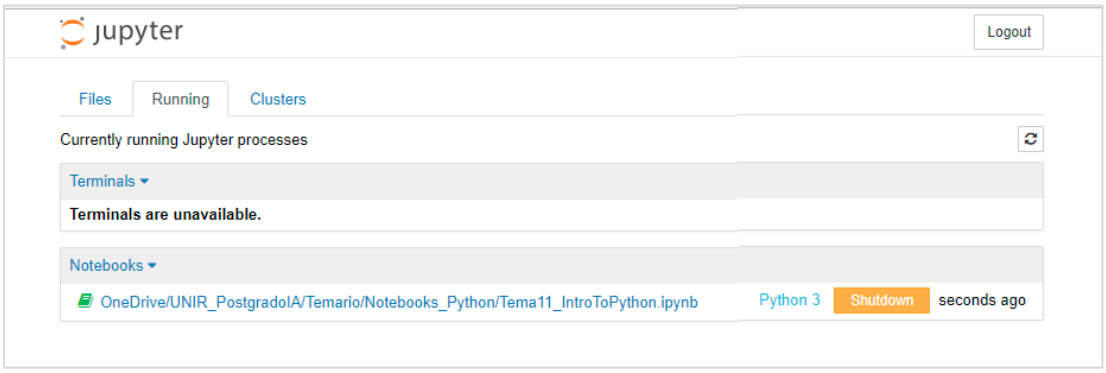

Figura 13. Notebooks activos (Jupyter).

Se recomienda encarecidamente cerrar los notebooks correctamente de cara a optimizar los recursos del sistema. Para ello y al final del menú Files encontramos la opción Close and Halt.

En los notebooks Jupyter introducimos el código, texto e imágenes en celdas. A través del botón + de la barra de comandos y del menú Insert podemos insertar tantas celdas como queramos. Existen varios tipos de celdas, el tipo más evidente es el tipo código. Otras opciones son Markdown y Formato en bruto (accede a las distintas opciones a través de Cell > Cell Type).

El formato Markdown es sumamente interesante para introducir las explicaciones oportunas que permitan entender mejor el trabajo realizado.

Los siguientes enlaces son muy útiles y recomendables para comprender de forma completa el abanico de funcionalidades de estos notebooks:

http://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html

https://jupyter.brynmawr.edu/services/public/dblank/Jupyter%20Notebook%20Users%20Manual.ipynb

Los notebooks se guardan de forma automática cada cierto tiempo. En cualquier caso y para asegurarnos que todo está correcto antes de cerrar, podemos guardar manualmente el notebook.

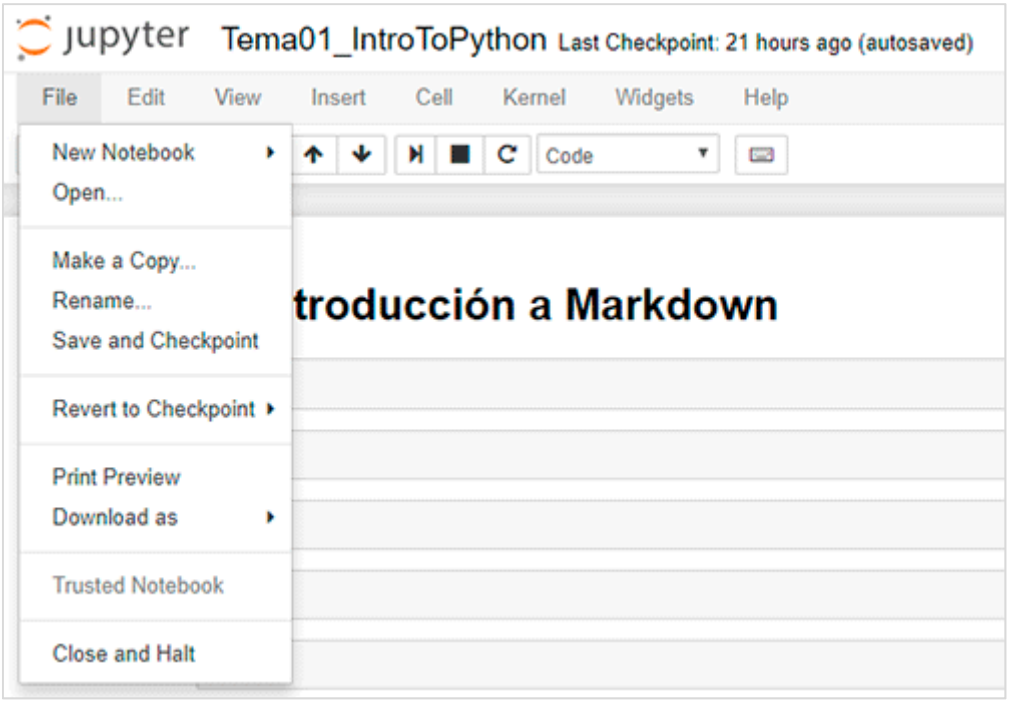

Figura 14. Guardar los últimos cambios en el notebook (Jupyter).

Al cerrar Anaconda Navigator, el servidor de Jupyter dejará de prestar servicios.

1.3. Variables y tipos de datos en Python

A la hora de trabajar con Python, debemos siempre considerar la referencia oficial disponible en: https://docs.python.org/3/reference/index.html

Una **variable** hace referencia a un lugar etiquetado en la memoria del equipo donde el usuario puede guardar, modificar y acceder a información. En caso de que el valor guardado sea fijo, se denomina **constante**.

Las variables tienen un tipo de datos concreto, numérico, cadena de texto, lista, etc. En el caso de Python y a diferencia de otros lenguajes, no es necesario indicar el tipo

de datos de la variable, puesto que Python elige el tipo de datos que corresponde en función del valor pasado. El nombre de las variables es sensible al uso de letras mayúsculas.

Las siguientes cadenas no pueden ser empleadas como nombres de variables, ya que corresponden con palabras reservadas de Python que tienen un significado especial: and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, is, in, import, lambda, not, or, pass, print, try, or, raise, return, while, with, yield.

Antes de comenzar con unos ejemplos básicos, debemos resaltar la enorme importancia que tiene para el programador ser **organizado, limpio y consistente** a la hora de escribir código. Es crucial emplear nombres correctos y explicativos a la hora de nombrar funciones y variables, así como estandarizar nuestras prácticas.

Por ejemplo, debemos evitar nombres de variables como «i», «j», «aux», etc. Estos nombres no son identificativos, hacen que el código sea más difícil de depurar y de comprender. En su lugar debemos emplear nombres descriptivos como «nombre_cliente», «calcula_ingresos_medios», «comprueba_datos»…

Por otro lado, debemos evitar caracteres propios del castellano como la letra ñ o las tildes.

Se pueden obtener más detalle consultando el libro de estilo de Google para programación en Python:
https://google.github.io/styleguide/pyguide.html

Es indispensable introducir los comentarios adecuados que permitan comprender lo que se está haciendo. Los comentarios de una línea vienen precedidos del símbolo #. Los comentarios de varias líneas emplean una triple comilla doble.

A continuación, algunos ejemplos:

```
# Esto es un comentario de una línea
""" Esto es un comentario
de varias lineas
"""

# Python solo reconoce como comentarios los textos que estén precedidos del carácter almohadilla
# El segundo caso se trata realmente de una cadena de texto de varias líneas que no se ha
# asignado a ninguna variable
```

Figura 15. Comentarios en Python.

```
# Constantes: imprimimos y comprobamos los resultados
print("Esto es una constante numérica: " + str(12345))
print("Esto es una cadena de texto: " + "hola que tal")
```

```
Esto es una constante numérica: 12345
Esto es una cadena de texto: hola que tal
```

```
# Variable numérica
variable_numerica = 3

# Variable cadena de texto (string)
variable_string = "hola"

# Imprimimos y comprobamos los resultados
# Como puedes observar, usamos el símbolo + para concatener cadenas
# Esto es un ejemplo básico de lo que se conoce como sobrecarga de operadores
# un mismo símbolo tiene distintos significados según el tipo a los que aplica
print("Esto es una variable numérica: " + str(variable_numerica))
print("Esto es una cadena de texto: " + variable_string)
```

Figura 16. Constantes y variables (Python).

```
# Los nombres son sensibles a mayúsculas
Variable = 5
variable = 2
print(Variable)
print(variable)
```

```
5
2
```

```
# Si queremos declarar un valor constante para todo el programa
# pero no queremos introducir el valor exacto de forma reiterativa
# podemos hacer así
CONSTANTE_PRUEBA = 5
# De esta forma, al estar en mayúscula indicamos que queremos que
# tenga tratamiento de constante. Si quisiéramos cambiar el valor
# bastaría con hacerlo solo en la declaración y no tener que ir
# reemplazando todos los 5 que hemos puesto
```

Figura 17. Nombres sensibles a mayúsculas y valores constantes (Python).

Podemos saber el tipo asociado a una variable concreta con la función Type. También hay funciones que nos permiten realizar conversión de tipos, como str (de número a cadena de texto), int (a entero), float (a número real):

```
# Trabajando con tipos
# ¿De qué tipo es variable?
print(type(variable))
# ¿y variable_string?
print(type(variable_string))
```

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

```
# Conversiones de tipos
numero_real = 3.45
print(numero_real)
print(str(numero_real))
print(type(numero_real))
print(type(str(numero_real)))
print(int(numero_real))
print(float(str(numero_real)))
print(int("5"))
```

```
3.45
3.45
<class 'float'>
<class 'str'>
3
3.45
5
```

Figura 18. Trabajando con tipos distintos (Python).

Tipos de datos complejos

Existen tipos de datos con una organización más compleja, tal es el caso de tuplas, listas y conjuntos. Además, las cadenas de caracteres admiten un tipo de manejo particular. Veamos estos aspectos con ejemplos prácticos.

```
# Acceso a elementos de una cadena mediante la posición
cadena = "Esto es una cadena"
print(cadena[5]) # ten en cuenta que el índice comienza por 0
print(cadena[5:9]) # rango de valores - de 5 a 9
print(cadena[:5]) # hasta 5
print(cadena[5:]) # de 5 en adelante
print(len(cadena)) # longitud
```

```
e
es u
Esto
es una cadena
18
```

Figura 19. Tratando cadenas de texto (Python).

Los conjuntos son colecciones no ordenadas y sin elementos repetidos. Pueden ser muy útiles en distintos tipos de situaciones.

```
# Conjuntos en Python
gastro = {'paella', 'cocido', 'fabada', 'empanada', 'jamon'}
print(gastro)
print(type(gastro))
# elementos del conjunto
print(next(iter(gastro)))
print(gastro.pop())
print(gastro)
# Ohhhh, paella ha desaparecido
gastro.add("jamon")
print(gastro)
# claro! un conjunto no tiene elementos repetidos
```

```
{'paella', 'jamon', 'cocido', 'fabada', 'empanada'}
<class 'set'>
paella
paella
{'jamon', 'cocido', 'fabada', 'empanada'}
{'jamon', 'cocido', 'fabada', 'empanada'}
```

Figura 20. Conjuntos básicos en Python

Las listas son una de las estructuras más frecuente en Python, permite acumular y manipular datos de forma sencilla.

```python
# trabajando con listas
gastro = ['paella', 'cocido', 'fabada', 'empanada', 'jamon']
# tipo?
print(type(gastro))

# asigna una variable a la lista
comer_en_esp = set(gastro)
print(comer_en_esp)
print(type(comer_en_esp))

# Condicionales
en_asturias = 'fabada' in comer_en_esp
print(en_asturias)

# escribe gastro. y pulsa sobre el tabulador para
# ver las distintas opciones disponibles
gastro.insert(len(gastro), "lentejas") # insertar al final
print(gastro)
# ¿Cómo va el orden aquí?
print(gastro.index("paella"))
# Parece ser que empieza por 0
```

```
<class 'list'>
{'paella', 'jamon', 'cocido', 'fabada', 'empanada'}
<class 'set'>
True
['paella', 'cocido', 'fabada', 'empanada', 'jamon', 'lentejas']
0
```

Figura 21. Listas en Python.

Por último, mencionaremos las tuplas, una tupla es una lista inmutable.

```python
# Tuplas
# Comprender qué es una tupla
tupla = [1, 2, 3, 4, 5]
print(tupla)
tupla[0] = 99
print(tupla) # por eso esto no es una tubla

tupla2 = {1, 2, 3, 4, 5}
print(tupla2)
tupla2[0] = 99 # Ohhh, algo ha ido mal
# por esto tupla2 sí es una tupla
```

```
[1, 2, 3, 4, 5]
[99, 2, 3, 4, 5]
{1, 2, 3, 4, 5}

---------------------------------------------------------------
TypeError                                 Traceback (most rec
<ipython-input-28-6998ba7d25d9> in <module>()
      8 tupla2 = {1, 2, 3, 4, 5}
      9 print(tupla2)
---> 10 tupla2[0] = 99 # Ohhh, algo ha ido mal
     11 # por esto tupla2 sí es una tupla

TypeError: 'set' object does not support item assignment
```

Figura 22. Tuplas en Python.

Otra estructura de datos interesante es el diccionario. Un diccionario permite identificar a sus elementos a través de una clave. La siguiente imagen muestra un ejemplo típico de uso.

```
# Diccionarios
diccionario = {'nombre' : 'Juan', 'edad' : 22, 'cursos': ['Python','Machine Learning','R']}

# Accediendo a los valores
print(diccionario['nombre'])
print(diccionario['edad'])
print(diccionario['cursos'])

# Recorrido modo lista
print(diccionario['cursos'][0])
# Recorrer todo el diccionario
for key in diccionario:
    print(key, ":", diccionario[key])
```

```
Juan
22
['Python', 'Machine Learning', 'R']
Python
nombre : Juan
edad : 22
cursos : ['Python', 'Machine Learning', 'R']
```

Figura 23. Diccionarios en Python.

Como estructura particular de Python, los diccionarios tienen sus propios métodos.

```
# Diccionarios
diccionario = {'nombre' : 'Juan', 'edad' : 22, 'cursos': ['Python','Machine Learning','R']}

# Accediendo a los valores
print(diccionario['nombre'])
print(diccionario['edad'])
print(diccionario['cursos'])

# Recorrido modo lista
print(diccionario['cursos'][0])
# Recorrer todo el diccionario
for key in diccionario:
    print(key, ":", diccionario[key])
```

```
Juan
22
['Python', 'Machine Learning', 'R']
Python
nombre : Juan
edad : 22
cursos : ['Python', 'Machine Learning', 'R']
```

Figura 24. Ejemplo de métodos de diccionarios (Python).

1.4. Condiciones, bucles y funciones en Python

Los ejemplos que veremos aquí constan de varias líneas, es imprescindible organizar correctamente el texto para que el compilador pueda interpretar el código. Python no usa las llaves ({, }) para indicar el comienzo y final de una instrucción condicional, bucle o función. En su lugar se guía por espacios en blanco. Así pues, debemos respetar los espacios en blanco. **Nunca mezclar tabuladores y espacios en blanco** pues es una fuente frecuente de errores.

Una instrucción condicional permite realizar una acción u otra en base a una condición previa.

Un **bucle** es una instrucción iterativa que permite realizar un conjunto de acciones un número concreto de veces. Las **funciones** son esenciales en cualquier lenguaje de programación, permiten encapsular un comportamiento común facilitando la reusabilidad e interpretabilidad del código.

Las siguientes imágenes muestran ejemplos que ayudarán a entender los conceptos planteados. Se recomienda a los alumnos repetir este código en sus propios entornos.

```
# Esto es una instrucción condicional
numero = 12
# % representa módulo o resto de una división
if (numero % 2 == 0):
    print("El número es par")
else:
    print("El número es impar")
```

```
El número es par
```

```
# Otra forma
print("El número es par") if numero % 2 == 0 else print("El número es impar")
```

```
El número es par
```

Figura 25. Condicionales en Python.

```
# Probemos ahora un bucle
for valor in range(5):
    print("Iteración número " + str(valor))
```

```
Iteración número 0
Iteración número 1
Iteración número 2
Iteración número 3
Iteración número 4
```

Figura 26. Ejemplo de un bucle en Python.

```
# bucle while
valor = 0
while (valor < 5):
    print("Iteración número " + str(valor))
    valor = valor + 1
# fin bucle
```

```
Iteración número 0
Iteración número 1
Iteración número 2
Iteración número 3
Iteración número 4
```

Figura 27. Bucle while en Python.

```
# Función para saber si un número es par o impar
# Recibe un parémetro de entrada para actuar en consecuencia
def es_par(numero):
    if (numero % 2 == 0):
        print("El número " + str(numero) + " es par")
    else:
        print("El número " + str(numero) + " es impar")
# Aquí acaba la función
# La indentación es muy importante

# Llamamos a la función como sigue
es_par(15)
es_par(12)
```

```
El número 15 es impar
El número 12 es par
```

Figura 28. Funciones en Python.

Una función puede devolver valores. Y pueden ser uno o varios valores.

```python
# Crearemos ahora una función que devuelve un valor
# Ejemplo sencillo, si es par multiplicamos por 2
# Si es impar sumamos dos
def transforma_numero(numero):
    if (numero % 2 == 0):
        return(numero * 2)
    else:
        return(numero + 2)
# Aquí acaba la función

# Llamamos a la función como sigue
prueba1 = transforma_numero(15)
prueba2 = transforma_numero(12)

print(str(prueba1) + "---" + str(prueba2))
```

```
17---24
```

Figura 29. Función que devuelve valores (Python).

```python
# Ejemplo de función que devuelve varios valores
# Similar a la anterior pero supongamos que ahora
# devuelve siempre número en primer
def transforma_numero(numero):
    if (numero % 2 == 0):
        return(numero, numero * 2)
    else:
        return(numero, numero + 2)
# Aquí acaba la función

# Llamamos a la función como sigue
numero1, prueba1 = transforma_numero(15)
numero2, prueba2 = transforma_numero(12)

print(str(numero1) + "_" + str(prueba1) +
      "---" + str(numero2) + "_" + str(prueba2))
```

```
15_17---12_24
```

Figura 30. Función que devuelve varios valores (Python).

1.5. Objetos en Python

En el paradigma de la programación orientada a objetos, una **clase** permite encapsular de forma conjunta atributos (datos) y métodos (acciones). Cuando una clase se instancia, da lugar a un objeto. Por ejemplo y dada la clase «coche», podemos crear a partir de ella una instancia objeto que sea un modelo de coche determinado con una matrícula concreta. Gracias a los atributos podemos saber el estado de un objeto. Los métodos de la clase permiten al objeto realizar acciones y comunicarse con su entorno.

Una explicación detallada del paradigma de la programación orientada a objetos excede el ámbito del presente tema y asignatura. Por tanto, nos centraremos en explicar un ejemplo típico de cómo crear un objeto en Python.

```
# clase rectángulo
class Rectangulo:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    descripcion = "esto es un rectangulo"
    autor = "anonimo"
    def area(self):
        return self.x * self.y
    def perimetro(self):
        return 2 * self.x + 2 * self.y
    def descripcion(self,text):
        self.description = text
    def nombreAutor(self,text):
        self.author = text
    def cambiaDimensiones(self,scale):
        self.x = self.x * scale
        self.y = self.y * scale

#creating objects
rectangulo1 = Rectangulo(90, 35)
rectangulo2 = Rectangulo(20, 11)

#describing the rectangles
rectangulo1.descripcion("Rectangulo de juguete")
print(rectangulo1.descripcion)
print(rectangulo2.area())
```

Figura 31. Clases y objetos en Python.

1.6. Introducción a la visualización con Python

Matplotlib es una de las librerías más usadas en Python para realizar visualizaciones sencillas. Veamos algunos ejemplos de cara a comprender el funcionamiento de esta herramienta. El código es autoexplicativo gracias a los comentarios.

La referencia completa se puede consultar en el siguiente enlace:

https://matplotlib.org/index.html

```
# Un gráfico simple
# Generate a normal distribution, center at x=0 and y=5
N_points = 100
x = np.random.randn(N_points)
y = .4 * x
plt.plot(x, y)
plt.grid()
```

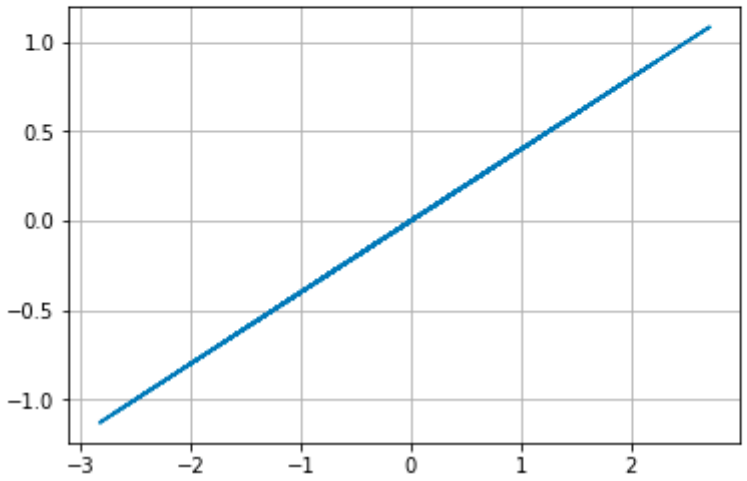

Figura 32. Gráfico simple en Python.

```python
# Algo más complejo
# fuente:
# https://matplotlib.org/devdocs/gallery/statistics/histogram_features.html
import numpy as np
import matplotlib.pyplot as plt
# semilla de números aletorios para
# poder reproducir el resultado
np.random.seed(1234)

# generamos nuestros propios datos de ejemplo
mu = 100  # media
sigma = 15  # desviación típica
x = mu + sigma * np.random.randn(437)

num_bins = 50 # intervalos
# observar que esta función devuelve dos valores
# ejecutar solo hasta aquí y comprobar si sale
# el histograma o no
fig, ax = plt.subplots()

# el histograma
n, bins, patches = ax.hist(x, num_bins, density=1)

# función de densidad
y = ((1 / (np.sqrt(2 * np.pi) * sigma)) *
     np.exp(-0.5 * (1 / sigma * (bins - mu))**2))
ax.plot(bins, y, '--')
ax.set_xlabel('valores x')
ax.set_ylabel('Función de densidad')
ax.set_title(r'Histograma: $\mu=100$, $\sigma=15$')

# Modificamos el espacio para impedir solapamientos
fig.tight_layout()
plt.show()
```

Figura 33. Histograma con función de densidad (Python).

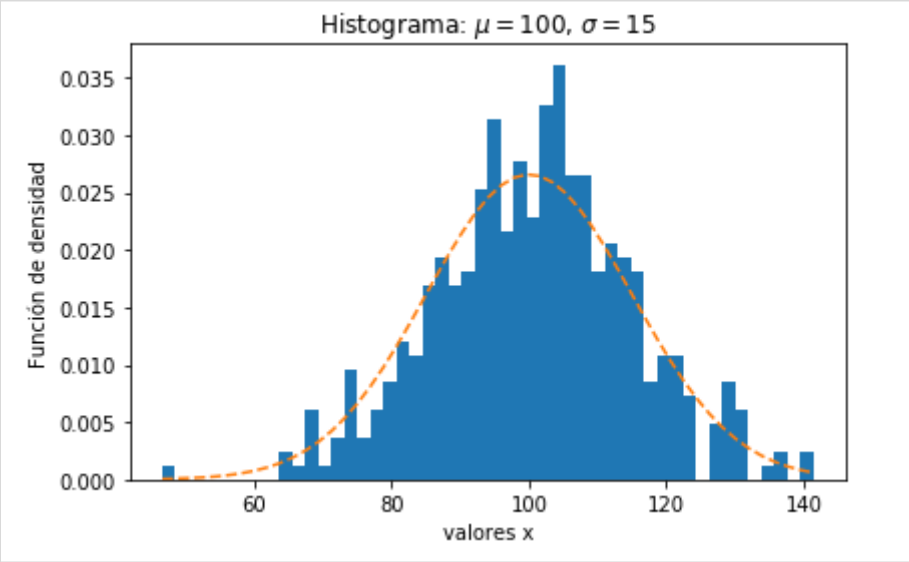

Figura 34. Resultado del código anterior (Python).

En la siguiente dirección web se pueden encontrar otros ejemplos sencillos con los que se recomienda trabajar:
https://matplotlib.org/tutorials/introductory/sample_plots.html#sphx-glr-tutorials-introductory-sample-plots-py

1.7. Referencias bibliográficas

McKinney, W. (2017). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython* (2ª ed.). Estados Unidos: O'Reilly Media.

+ Información

A fondo

Introduction to Programming

En esta web puedes refrescar conocimientos básicos sobre variables, funciones, bucles, instrucciones condicionales, etc.

Accede al contenido a través del aula virtual o desde la siguiente dirección web:
https://en.wikiversity.org/wiki/Introduction_to_Programming

Python Introduction

https://developers.google.com/edu/python/

Introducción a la programación con Python de Google for Education. Un recurso complementario a lo que se ha explicado en el tema, muy útil para profundizar.

Accede a la Google Python Class a través del aula virtual o desde la siguiente dirección web:
https://developers.google.com/edu/python/

Test

1. Python es un lenguaje:

A. Compilado y gratuito.

B. Interpretado y de pago.

C. Compilado y de pago.

D. Interpretado y gratuito.

2. Respecto a Python 2.x, Python 3.x:

A. Solo cambia el logotipo.

B. Es más eficiente y reusable.

C. Es más eficiente pero mucho más complejo de usar.

D. Es más reusable pero pierde notablemente en eficiencia.

3. Para trabajar con Python:

A. Podemos trabajar mediante consola.

B. Podemos trabajar mediante entornos de desarrollo como Spyder.

C. Podemos trabajar mediante notebooks.

D. Todas las anteriores.

4. A la hora de trabajar con Jupyter:

A. Debemos cerrar correctamente los notebooks que no estemos usando de cara a optimizar los recursos del sistema.

B. No necesitamos cerrar los notebooks. Jupyter dispone de un recolector de basura similar a Java que cierra lo que no esté en uso.

C. Debemos salvar continuamente el notebook manualmente para no perder los cambios introducidos.

D. Primero debemos probar el código en un IDE o por consola.

5. A la hora de programar con Python:

A. Todas las variables deben empezar por un número.

B. Los nombres de variables no son sensibles a mayúsculas.

C. Se deben seguir criterios fijos de identificación del código o no funcionará correctamente.

D. Las llaves para indicar bloques de código son opcionales.

6. Los comentarios en Python:

A. Son importantes y necesarios de cara a facilitar la comprensión del código.

B. No son necesarios.

C. Solo sirven para poner notas a otros compañeros que vayan a trabajar con él código.

D. Si no se usan comentarios, el programa no funcionará.

7. Respecto a los conjuntos podemos decir que:

A. Admiten elementos repetidos.

B. El orden es importante.

C. Python no admite conjuntos.

D. No admiten elementos repetidos.

8. Una tupla:

A. Es lo mismo que un conjunto.

B. Es exactamente lo mismo que una lista.

C. Admite modificar elementos.

D. Es una lista inmutable.

9. Matplotlib:

A. Es una librería de cálculo matricial.

B. Es una librería de cálculo simbólico.

C. Es una librería de visualización.

D. Es una librería de *machine learning*.

10. Para dibujar un histograma en Python, llamamos a la función:

A. plotHistogram().

B. pltHistogram().

C. hist().

D. subplot().