Introducción rápida a Python

Inteligencia Artificial 2020/2021

Belén Díaz Agudo

Algunos de los ejemplos mostrados son del curso de Introducción a la programación en Phyton de los cursos CFI https://cursosinformatica.ucm.es/ (https://cursosinformatica.ucm.es/)

También se ha utilizado material de introducción a Phyton del profesor Jose Luis Reina de la universidad de Sevilla.

Tenéis que ejecutar cada celda de código y observar el resultado para comprender el funcionamiento de las instrucciones en la celda. Observar la sintaxis del lenguaje python para cada tipo de instrucción.

Expresiones y variables

Las variables en Python no hay que declararlas

Una variable es una **referencia** a una posición de memoria, en la que está almacenada el dato.

In [1]:
```
(2+3)**4
(50-5*6)/4
# Solo se muestra el resultado de la última instrucción
```

Out[1]: 5.0

In [2]:
```
x=(2+3)**4
x
```

Out[2]: 625

In [3]:
```
ancho = 20
alto = 5*9
area = ancho * alto
```

In [4]:
```
ancho, alto, area
```

Out[4]: (20, 45, 900)

In [5]:
```
area*=2
```

In [6]:
```
x = 1
y = 2
print('suma:', x + y)
print('resta:', x - y)
print('multiplicación:', x * y)
print('division:', x / y)
print('division entera:', x // y)
print('potencia:', y ** 10)
```

```
suma: 3
resta: -1
multiplicación: 2
division: 0.5
division entera: 0
potencia: 1024
```

In [7]:
```
x = 5
x += 2
x  # El último valor de la celda se imprime por defecto
```

Out[7]: 7

In [29]:
```
print(float(7))
```

```
7.0
```

Cadenas de caracteres (Strings)

Son secuencias de caracteres entre comillas simples o dobles.

In [8]:
```
c1= "Esto es una cadena "
```

In [9]:
```
c2=' y esto tambien es una cadena'
```

In [10]:
```
frase = c1 + c2
```

In [11]:
```
frase
```

Out[11]: 'Esto es una cadena y esto tambien es una cadena'

In [12]:
```
# operaciones con cadenas
mayusculas = 'antonio'.upper()
mayusculas = 'se queda con el ultimo'.upper()
print(mayusculas)
cadena_datos = '{} tiene {} años'.format('juan', 21)
print(cadena_datos)
concatenar = 'hola' + ' ' + 'mundo!'
print(concatenar)
```

```
SE QUEDA CON EL ULTIMO
juan tiene 21 años
hola mundo!
```

In [13]:
```
concatenar+=" y fin.."
```

In [14]:
```
concatenar
```

Out[14]:
```
'hola mundo! y fin..'
```

In [15]:
```
frase = concatenar
frase*2
```

Out[15]:
```
'hola mundo! y fin..hola mundo! y fin..'
```

In [16]:
```
frase
```

Out[16]:
```
'hola mundo! y fin..'
```

Se puede acceder a caracteres concretos de un string, mediante su índice de posición. En Python los índices *empiezan a contar en 0* y pueden ser negativos (en ese caso, considera el string como circular y cuenta hacia atrás desde el último)

In [17]:
```
frase[7]
```

Out[17]:
```
'n'
```

In [18]:
```
frase[-1]
```

Out[18]:
```
'.'
```

In [19]:
```
frase[3]+frase[-4]
```

Out[19]:
```
'ai'
```

In [20]:
```
#Operador de slicing en Python: dada una secuencia l (como por ejemplo un strin
g), la notación l[inicio:fin] indica la subsecuencia de l que comienza en la posi
ción de índice inicio y acaba en la posición anterior a fin.
frase[2:6]
```

Out[20]:
```
'la m'
```

In [21]:
```
#En la operación de _slicing_ se puede incluir un tercer argumento `l[inicio:fin:
salto]`,
# indicando el salto a la hora de recorrer la lista.
#El salto puede ser negativo, indicando recorrido desde el final.
frase[2:7:2]
```

Out[21]:
```
'l u'
```

In [22]:
```
# Por defecto inicio es 0 y fin es la última posicion de la secuencia y el salto
 es 1.
#Si el `salto` es negativo, los valores por defecto de `inicio` y `fin` se interc
ambian.
#Es decir, si no se da `inicio`, sería la última posición, y si no se da `fin` se
ría la primera
frase[0::-1]
frase[::-1]
```

Out[22]:
```
'..nif y !odnum aloh'
```

El tipo de dato string es una clase predefinida, y las cadenas de caracteres concretas son objetos de la clase string. Por tanto a un objeto de la clase string se le pueden aplicar los métodos que están predefinidos para la clase string.

In [23]:
```
cad="En un lugar de La Mancha"
```

In [24]:
```
#el método index de la clase String busca la posicion en la que está un substring en la cadena
# y da error si no lo encuentra
cad.index("Mancha")
```

Out[24]:
```
18
```

In [25]:
```
cad.index("mancha")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-25-b13daaf55a8f> in <module>
----> 1 cad.index("mancha")

ValueError: substring not found
```

Otros métodos de la clase String son find, upper, count, join, split,.. El resultado de join y split son listas que aparecen entre corchetes y separadas por ,

In [26]:
```
"Rojo y blanco y negro".split(" ")
```

Out[26]:
```
['Rojo', 'y', 'blanco', 'y', 'negro']
```

Print y format: la función print permite escribir cadenas de caracteres por pantalla, y el método format de la clase string nos permite manejar cadenas de caracteres que contienen ciertos "huecos" (*templates*) que se rellenan con valores concretos (al estilo de C)

In [27]:
```
uno =" la "
dos =" es "
print(uno,"Inteligencia",dos,"Artificial")
```

```
 la  Inteligencia  es  Artificial
```

In [ ]:
```
# La cadena c tiene 3 huecos
c="{0} por {1} es {2}"
x,y,u,z = 2,3,4,5
# format le pasa los parámetros para los 3 huecos
print(c.format(x,y,x*y))
print(c.format(u,z,u*z))
```

Pedir datos por consola

In [ ]:
```
nombre = input('Dime tu nombre:') # input siempre devuelve una cadena str()
numero_entero = int(input('Dime un número entero:'))
numero_real = float(input('Dime un número real:'))
print('Nombre:', nombre, 'Entero:', numero_entero, 'Real:', numero_real)
```

Condicionales

In [ ]:
```
x = 25
if x % 2 == 0:
    print('Divisible por 2')
elif x % 3 == 0:
    print('Divisible por 3')
elif x % 5 == 0:
    print('Divisible por 5')
else:
    print('Ni idea')
```

In [ ]:
```
# condiciones compuestas
year = int(input('Introduce un año:'))
if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):
    print('bisiesto')
else:
    print('no es bisiesto')
```

In [ ]:
```
2==2
```

Los valores logicos de verdad y falsedad en Python son `True` y `False`. Podemos comparar números con el operador de comparación `==` (no confundir con el de asignación `=`), o comprobar si son distintos con `!=`, tambien se pueden usar los operadores lógicos usuales.

In [ ]:
```
2!=4 and 2==1
```

Bucles básicos

In [ ]:
```
suma = 0
i = 1
while i <= 10:
    suma += i
    i += 1
suma
```

Tuplas

Las *tuplas* en Python son secuencias de datos separadas por comas. Usualmente van entre paréntesis, aunque no es obligatorio (excepto para la tupla vacía). Ejemplos:

In [1]:
```
# El operador coma crea tuplas. Se pueden escribir o no entre paréntesis.
a = 1, 2, 3
print(a)
b = (4, 5, 6)
print(b)
```

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

Como las tuplas son secuencias, algunas de las operaciones de strings, también se pueden aplicar a las tuplas. En particular, el acceso a elementos a través de la posición, el operador de slicing, o la concatenación

In [2]:
```
# Se pueden leer sus componentes de forma individual
a[0]
```

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

In [3]:
```
# Las tuplas son inmutables.
#a[0] = 10 # ERROR!
```

In [4]:
```
# La asignación entre tuplas asigna campo a campo.
# (parece asignación múltiple pero es asignación entre tuplas)
a, b, c = 10, 20, 30
print(a)
print(b)
print(c)
```

```
10
20
30
```

In [5]:
```
1, # No confundir con (1), que sería simplememente el número 1
```

Out[5]:
```
(1,)
```

In [6]:
```
(1)
```

Out[6]:
```
1
```

In [7]:
```
1
```

Out[7]:
```
1
```

In [8]:
```
a=("Uno","Dos","Tres","Cuatro")
```

In [9]:
```
a[::1]
```

Out[9]:
```
('Uno', 'Dos', 'Tres', 'Cuatro')
```

In [10]:
```
#inversión de una tupla
a[::-1]
```

Out[10]:
```
('Cuatro', 'Tres', 'Dos', 'Uno')
```

Las tuplas son tipos de datos **inmutables**. Esto significa que una vez creadas, no podemos cambiar su contenido.

In [11]:
```
a[1]="otro"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-0f9c0ecf27a0> in <module>
----> 1 a[1]="otro"

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

Rangos

In [12]:
```
range(1, 10) # Representa los números del 1 al 9  >=1 y <9
```

Out[12]:
```
range(1, 10)
```

In [13]:
```
# los bucles for permiten recorrer estructuras de datos
for x in range(1, 10):
    print(x)
```

```
1
2
3
4
5
6
7
8
9
```

In [14]:
```
# del 1 al 9 de 3 en 3
for x in range(1, 10, 3):
    print(x)
```

```
1
4
7
```

In [15]:
```
# del 10 al 2 hacia abajo
for x in range(10, 1, -1):
    print(x)
```

```
10
9
8
7
6
5
4
3
2
```

In [1]:
```
# si sólo tiene una argumento es el límite superior
for x in range(3):
    print(x)
```

```
0
1
2
```

Listas

Las listas, al igual que las tuplas, son **secuencias** de datos. Pero son **mutables** (es decir, podemos cambiar su contenido).

Una lista se representa como una secuencia de datos entre corchetes y separadas por comas.

In [16]:
```
tupla=("Uno","Dos","Tres","Cuatro")
lista=["Uno","Dos","Tres","Cuatro"]
```

In [17]:
```
lista[0]='Otro'
```

In [18]:
```
lista
```

Out[18]:
```
['Otro', 'Dos', 'Tres', 'Cuatro']
```

In [19]:
```
a = [] # Lista vacía
len(a) # Devuelve la longitud de una lista
```

Out[19]:
```
0
```

In [20]:
```
# Pueden contener cualquier tipo de elementos combinados.
# Se añaden elementos a la lista con append
a.append('hola')
a.append(123)
a.append(True)
a
```

Out[20]:
```
['hola', 123, True]
```

Los métodos `append` y `extend`, respectivamente añaden un elemento al final, y concatenan una lista al final. Nótese que son métodos **destructivos**, en el sentido de que modifican la lista a la que se aplican.

In [21]:
```
a = ["Hola"]
a.extend([6,"que tal"])
a
```

Out[21]:
```
['Hola', 6, 'que tal']
```

In [22]:
```
# Podemos acceder y modificar sus elementos con acceso directo
a[1] += 1
a
```

Out[22]:
```
['Hola', 7, 'que tal']
```

In [23]:
```
# podemos borrar elementos a partir de su posición
del a[0]
a
```

Out[23]:
```
[7, 'que tal']
```

In [24]:
```
# podemos borrar elementos. Si el elemento no está o la lista está vacía da erro
r.
a.remove(7)
a
```

Out[24]:
```
['que tal']
```

In [25]:
```
# El método pop elimina un elemento de una lista (especificando la posición, por
 defecto la última), y devuelve dicho elemento como valor.
# Si hacemos pop en una lista vacía da error
a.pop()
```

Out[25]:
```
'que tal'
```

In [26]:
```
# Podemos crear listas con elementos
a = [1, 2, 3, 4]
a
```

Out[26]:
```
[1, 2, 3, 4]
```

In [27]:
```
# En python podemos definir listas por 'comprensión':
a = [2*x for x in range(1,10)]
a
```

Out[27]:
```
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

In [28]:
```
# el bucle for permite recorrer listas
for elemento in a:
    print(elemento)
```

```
2
4
6
8
10
12
14
16
18
```

In [29]:
```
a = list(range(1,11)) # el constructor list() crea listas
a
```

Out[29]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Con el operador de slicing ':' que hemos visto para tuplas podemos seleccionar trozos de una lista y trabajar con ellos como si fueran listas. En realidad son vistas sobre la lista original, no se duplican los elementos en memoria.

In [30]:
```
#El método `insert` inserta un elemento en una lista, en una posición dada
a.insert(3,"x")
```

In [31]:
```
a
```

Out[31]:
```
[1, 2, 3, 'x', 4, 5, 6, 7, 8, 9, 10]
```

In [32]:
```
# Usa el valor por defecto de la posicion inicial
a[:2]
```

Out[32]:
```
[1, 2]
```

In [33]:
```
a[0:3] # selección de los 3 primeros elementos. Tambien se puede escribir a[:3]
```

Out[33]:
```
[1, 2, 3]
```

In [34]:
```
a[1:4] # selección de los elementos en las posiciones 1, 2 y 3
```

Out[34]:
```
[2, 3, 'x']
```

In [35]:
```
a[1:6:2] # saltando de 2 en 2
```

Out[35]:
```
[2, 'x', 5]
```

In [36]:
```
a[3:0:-1] # Elementos en posiciones 3, 2, 1
```

Out[36]:
```
['x', 3, 2]
```

In [37]:
```
a[::-1] # ver la lista al revés
```

Out[37]:
```
[10, 9, 8, 7, 6, 5, 4, 'x', 3, 2, 1]
```

In [38]:
```
a[-3:] # ultimos 3 elementos de la lista
```

Out[38]:
```
[8, 9, 10]
```

In [39]:
```
bocadillo = ["pan", "jamon", "pan"]
bocadillo = 2*bocadillo[:2] + ["huevo"]
print(bocadillo)
bocadillo + [bocadillo[-1]]
```

```
['pan', 'jamon', 'pan', 'jamon', 'huevo']
```

Out[39]:
```
['pan', 'jamon', 'pan', 'jamon', 'huevo', 'huevo']
```

In [40]:
```
"tomate" in bocadillo
```

Out[40]:
```
False
```

In [41]:
```
# Un error muy común debido a que las variables en Phyton son referencias.

l=[28,1,54,6]
m=l # asignamos a m "el valor" de l
m[2]=11 # cambiamos m
```

La secuencia anterior tiene un error muy común debido a que las variables en Phyton son referencias.Piensa qué pasaría si consultamos m y l

In [42]:
```
m
```

Out[42]:
```
[28, 1, 11, 6]
```

In [43]:
```
l
```

Out[43]:
```
[28, 1, 11, 6]
```

¿Se te ocurre alguna manera **correcta** de obtener una versión modificada de una lista sin cambiar la original?

In [44]:
```
l=[28,1,22,6]
m=l[:] # asignamos a m una COPIA del valor de l usando slicing
# Estamos haciendo una copia idéntica usando los valores por defecto de inicio y
 fin.
m[2]=11 # cambiamos m
```

In [45]:
```
m
```

Out[45]:
```
[28, 1, 11, 6]
```

In [46]:
```
l
```

Out[46]:
```
[28, 1, 22, 6]
```

Conjuntos

In [47]:
```
# definir conjuntos
dias_semana = { 'lunes', 'martes', 'miércoles', 'jueves', 'viernes', 'sábado'}
```

In [48]:
```
# pertenencia
'miércoles' in dias_semana
```

Out[48]: True

In [49]:
```
'domingo' in dias_semana
```

Out[49]: False

In [50]:
```
# añadir elementos
dias_semana.add('domingo')
'domingo' in dias_semana
```

Out[50]: True

In [51]:
```
# los elementos no se guardan ordenados
dias_semana
```

Out[51]: {'domingo', 'jueves', 'lunes', 'martes', 'miércoles', 'sábado', 'viernes'}

In [52]:
```
# no admite elementos repetidos
print(len(dias_semana)) # número de elementos
dias_semana.add('lunes')
print(len(dias_semana))
```

```
7
7
```

In [53]:
```
# recorrido de conjuntos
for elemento in dias_semana:
    print(elemento)
```

```
domingo
lunes
viernes
martes
miércoles
jueves
sábado
```

In [54]:
```
# el constructor set() crea conjuntos
# quitar elementos repetidos de una lista
set([1,2,3,1,2,3,1,2,3])
```

Out[54]: {1, 2, 3}

In [55]:
```
# También se pueden definir por comprensión
cuadrados = { x**2 for x in range(1,10)}
cuadrados
```

Out[55]: {1, 4, 9, 16, 25, 36, 49, 64, 81}

Diccionarios

Un diccionario en Python es una estructura de datos que permite asignar valores a una serie de elementos (claves). En otros lenguajes de programación, esta estructura de datos se conoce como map o tabla hash. Se representan como un conjunto de parejas clave:valor, separadas por comas y entre llaves. En el siguiente ejemplo, la clave "juan" tiene asignado el valor 4098, y la clave "ana"tiene asignado el valor 4139

In [56]:
```
dict_telefonos = {"juan": 4098, "ana": 4139}
#acceso a clave
dict_telefonos["ana"]
#añadir una nueva pareja clave/valor
dict_telefonos["pedro"]=2321
#cambiar un valor
dict_telefonos["ana"] = 4140
```

In [57]:
```
dict_telefonos
```

Out[57]:
```
{'juan': 4098, 'ana': 4140, 'pedro': 2321}
```

In [58]:
```
#borrar una pareja clave/valor
del dict_telefonos["ana"]
print(dict_telefonos)
```

```
{'juan': 4098, 'pedro': 2321}
```

In [59]:
```
# Contienen pares (clave, valor)
diccionario = {'uno': 'one', 'dos': 'two', 'tres': 'three', 'cuatro': 'four'}
diccionario
```

Out[59]:
```
{'uno': 'one', 'dos': 'two', 'tres': 'three', 'cuatro': 'four'}
```

In [60]:
```
# acceso a elementos por clave
diccionario['uno']
```

Out[60]:
```
'one'
```

In [61]:
```
# claves
diccionario.keys()
```

Out[61]:
```
dict_keys(['uno', 'dos', 'tres', 'cuatro'])
```

In [62]:
```
# pares (clave, valor)
diccionario.items()
```

Out[62]:
```
dict_items([('uno', 'one'), ('dos', 'two'), ('tres', 'three'), ('cuatro', 'fou
r')])
```

In [63]:
```
# Son mutables
diccionario['cinco'] = 'five'
diccionario
```

Out[63]:
```
{'uno': 'one',
 'dos': 'two',
 'tres': 'three',
 'cuatro': 'four',
 'cinco': 'five'}
```

In [64]:
```
# elementos
len(diccionario)
```

Out[64]:
```
5
```

In [65]:
```
# recorrido
for clave in diccionario:
    print(clave, diccionario[clave])
```

```
uno one
dos two
tres three
cuatro four
cinco five
```

In [66]:
```
for clave, valor in diccionario.items():
    print(clave, valor)
```

```
uno one
dos two
tres three
cuatro four
cinco five
```

Algo más sobre Bucles

In [67]:
```
# Buscar la posición ind de un elemento en una lista. Si no se encuentra, ind=-1

ind =0
busco = "premio"
lst = ["nada","pierdo","premio","sigue"]

while ind < len(lst) and lst[ind] != busco:
    ind += 1

if ind == len(lst):
    ind=-1

ind
```

Out[67]: 2

El bucle `for` :

- `for var in seq`
- `for var in range(n)`

En el primer caso, `seq` es una secuencia (por ejemplo, una lista, tupla, o string), generándose tantas iteraciones como elementos tenga la secuencia, y en cada iteración, `var` va tomando los sucesivos valores de la secuencia. Por ejemplo:

In [68]:
```
# Cálculo de media aritmética
l, suma, n = [1,5,8,12,3,7], 0, 0

for e in l:
    suma += e
    n +=1

suma/n
```

Out[68]: 6.0

In [69]:
```
# Cálculo de números primos entre 3 y 20

primos = []
for n in range(3, 20, 2):
    for x in range(2, n):
        if n % x == 0:
            print(n, "es", x, "*", n//x)
            break
    else:
        primos.append(n)

primos
```

```
9 es 3 * 3
15 es 3 * 5
```

Out[69]: [3, 5, 7, 11, 13, 17, 19]

Otros patrones de iteración

- `for k in dicc:` itera la variable `k` sobre las claves del diccionario `dicc`.
- `for (k,v) in dic.items():` itera el par `(k,v)` sobre los pares $(clave, valor)$ del diccionario `dicc`.
- `for (i,x) in enumerate(l):` itera el par `(i,x)`, donde `x` va tomando los distintos elementos de `l` e `i` la correspondiente posición de `x` en `l`.
- `for (u,v) in zip(l,m):` itera el par `(u,v)` sobre los correspondientes elementos de `l` y `m` que ocupan la misma posición.
- `for x in reversed(l):` itera `x` sobre la secuencia `l`, pero en orden inverso.

In [70]:
```
preguntas = ["nombre", "apellido", "color favorito"]
respuestas = ["Juan", "Pérez", "rojo"]

for p, r in zip(preguntas, respuestas):
    print("Mi {} es {}.".format(p, r))
```

```
Mi nombre es Juan.
Mi apellido es Pérez.
Mi color favorito es rojo.
```

Funciones

In [71]:
```
# Definición de una función
def resta(x, y):
    return x - y
```

In [72]:
```
resta(20, 5)
```

Out[72]: 15

In [73]:
```
# podemos pasar los algumentos por posición o por nombre
resta(y=2, x=10)
```

Out[73]: 8

In [74]:
```
# si mezclamos, los primeros tienen que pasarse por posición
resta(10, y=3)
```

Out[74]: 7

In [75]:
```
# Los parámetros de tipos básicos se pasan por valor
def incrementa_mal(x):
    x += 1

x = 5
incrementa_mal(x)
x
```

Out[75]: 5

In [76]:
```
# Los parámetros de tipos compuestos se pasan por variable
def add_dos(lista):
    lista.append(2)

l = []
add_dos(l)
l
```

Out[76]: [2]

In [77]:
```
# Las funciones sólo pueden devolver un valor (return) pero puede ser una tupla
 (!!!)
def suma_resta(x, y):
    return x + y, x - y

a, b, = suma_resta(10, 3)
a, b
```

Out[77]: (13, 7)

In [78]:
```
# Los parámetros pueden tener valores por defecto
def incrementa(x, delta=1):
    return x + delta

print(incrementa(5, 2))
print(incrementa(5))
```

```
7
6
```

Módulos

In [79]:
```
# Para usar las funciones de un módulo, primero tenemos que importarlo
import math

math.factorial(10)
```

Out[79]: 3628800

In [80]:
```
# Si el nombre del módulo es muy largo y no queremos escribirlo todo el rato
# podemos darle un 'apodo'
import numpy as np

np.add([1, 2, 3], [3, 4, 5]) # suma de vectores
```

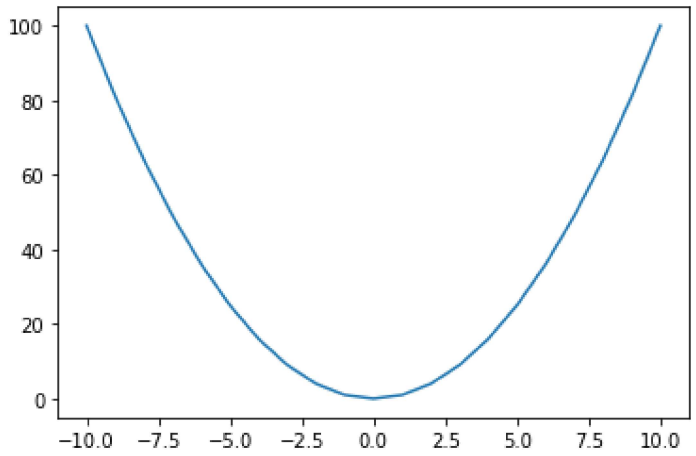
Out[80]:
```
array([4, 6, 8])
```

In [81]:
```
# También podemos importar funciones concretas y ahorranos escribir el nombre
# del módulo cuando las usemos
from math import factorial

factorial(10)
```

Out[81]:
```
3628800
```

In [82]:
```
# Las bibliotecas normalmente exportan varios módulos que pueden estar anidados
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 21) # 21 valores desde el -10 hasta el 10 equidistantes
y = [e*e for e in x]
plt.plot(x, y)
```

Out[82]:
```
[<matplotlib.lines.Line2D at 0x1de25ffbf10>]
```



Y muchas cosas más...

https://docs.python.org (https://docs.python.org)

Numpy

Una de las grandes carencias de Python es que no tiene arrays y las listas no son eficientes para trabajar con grandes cantidades de datos. Numpy es la biblioteca que permite la gestión eficiente de arrays multidimensionales.

In [83]:
```
import numpy as np

# convertir una lista en un array
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
a = np.array(l)
a
```

Out[83]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [84]:
```
# cambiar la forma de un array
b = a.reshape(3,3)
b
```

Out[84]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [85]:
```
# dimensiones
b.shape
```

Out[85]:
```
(3, 3)
```

In [86]:
```
# acceso a elementos
print(b[0,0])
print(b[0,1])
print(b[2,2])
```

```
1
2
9
```

In [97]:
```
# vistas de subarrays
b[0:2, 0:3]
```

Out[97]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [98]:
```
# arrays inicializados, te devuelve un nuevo array de 2 filas y 5 columnas inicia
lizado a 0
np.zeros((2,5))
```

Out[98]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [99]:
```
np.ones((2,5))
```

Out[99]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

In [100]:
```
# devuelve una matriz diagonal 5x5
np.eye(5,5)
```

Out[100]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

In [101]:
```
# arange devuelve un array con los valores del 1 al 9
m = np.arange(1,10).reshape(3,3)
m
```

Out[101]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [102]:
```
# Operaciones con todos los elementos
m * 2
```

Out[102]:
```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

In [103]:
```
2 ** m
```

Out[103]:
```
array([[  2,   4,   8],
       [ 16,  32,  64],
       [128, 256, 512]], dtype=int32)
```

In [104]:
```
# suma de todos los elementos
m.sum()
```

Out[104]:
```
45
```

In [105]:
```
# suma por columnas
m.sum(axis=0)
```

Out[105]:
```
array([12, 15, 18])
```

In [106]:
```
# suma por filas
m.sum(axis=1)
```

Out[106]:
```
array([ 6, 15, 24])
```

In [107]:
```
# producto vectorial de dos matrices
np.dot(m, m)
```

Out[107]:
```
array([[ 30,  36,  42],
       [ 66,  81,  96],
       [102, 126, 150]])
```

Y muchas cosas más...

http://www.numpy.org/ (http://www.numpy.org/)

In [ ]:

In [ ]: