Programación de alto nivel con python

Dr. Luis Gerardo de la Fraga

E-mail: fraga@cs.cinvestav.mx
http://cs.cinvestav.mx/~fraga

Departamento de Computación
Cinvestav

27 de septiembre, 2016

Contenido

1. ¿Por qué python?
2. Una breve introducción a Unix
3. Programando en python
4. Resolviendo problemas con python

¿Porqué aprender python?

- Es un lenguaje de alto nivel
- Es fácil construir programas rápidamente
- Es uno de los lenguajes más usandos
- Es el lenguaje sugerido para programar en RaspBerryPi

Lenguajes de alto nivel

- C es un lenguaje de nivel medio
- Los leguajes de alto nivel son interpretados
- No se definen tipos de datos
- Tienen una máquina de tiempo de ejecución (para uso de memoria con un recolector de basura)
- Se puede empotrar funciones en C dentro de Python si se necesita eficiencia.

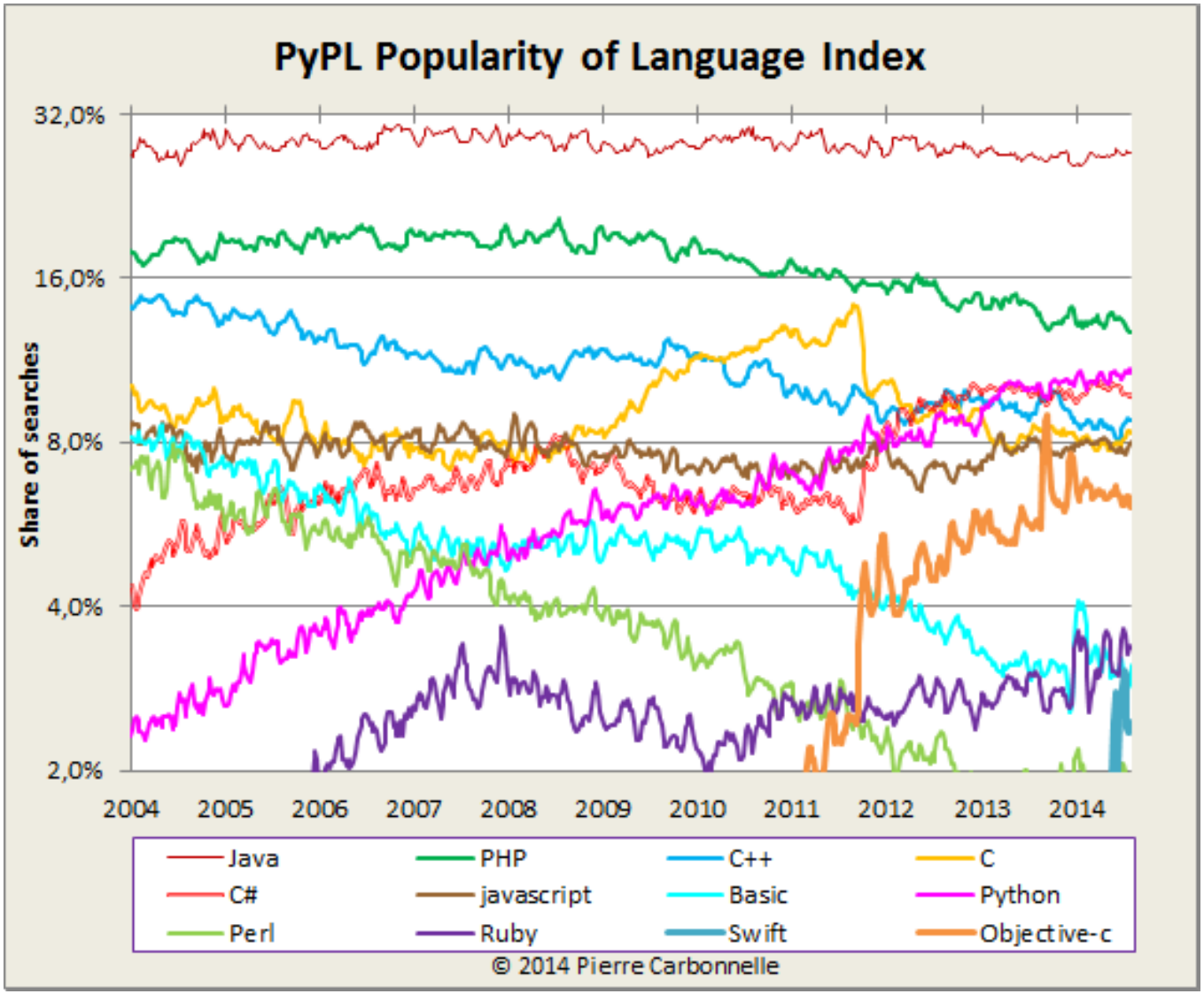
Prototipado rápido

- Son ideales para realizar programas que procesan texto y generan texto
- Para generación dinámica de páginas WEB
- Si se tiene una idea, puede obtenerse un programa funcional en minutos u horas

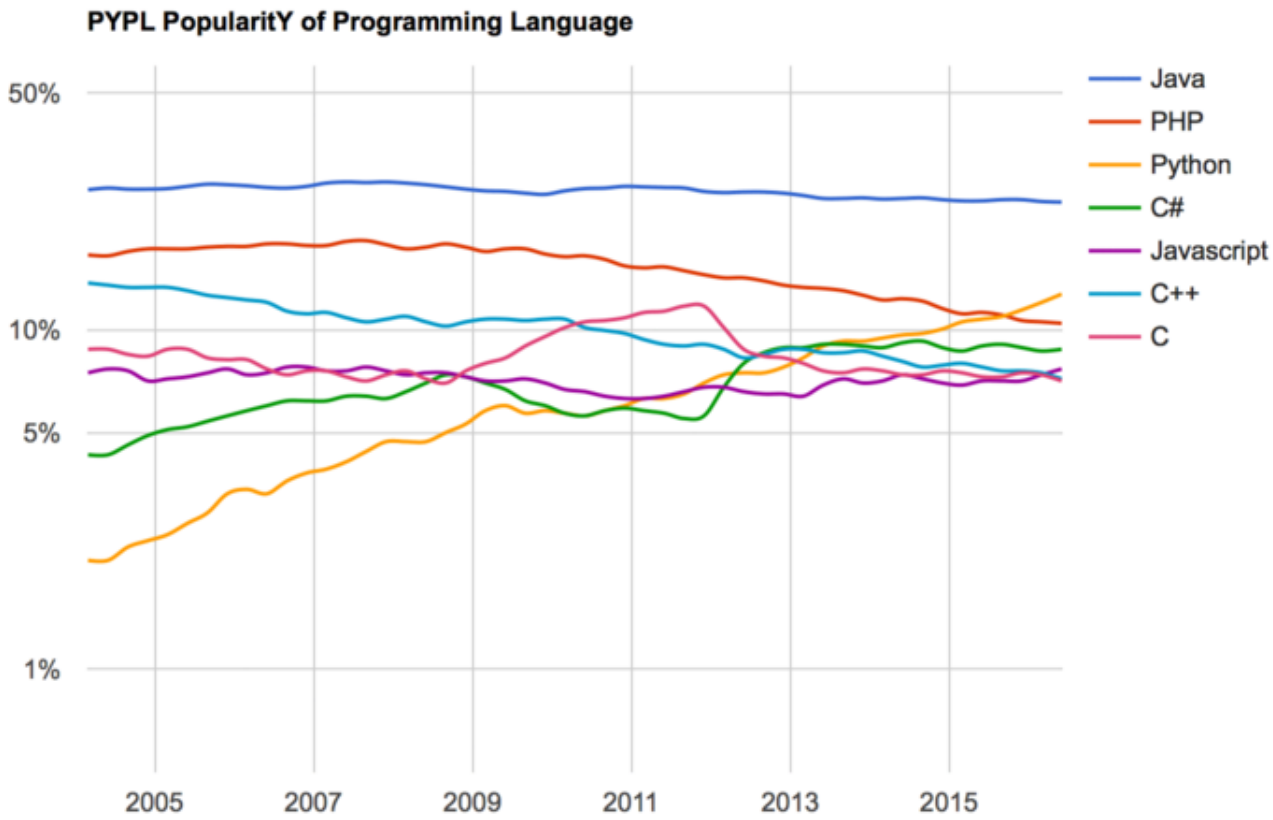
Índice de Popularidad de los Lenguajes de Programación

PYPL PopularitY of Programming Language index
`http://pypl.github.io/PYPL.html`

Posición	Cambio	Lenguaje	Porción	Tendencia
1		Java	23.9 %	-0.5 %
2	↑	Python	12.8 %	+2.1 %
3	↓	PHP	10.5 %	-0.8 %
4		C#	8.8 %	-0.5 %
5	↑↑	Javascript	7.7 %	+0.6 %
6	↓	C++	7.2 %	-0.4 %
7	↓	C	7.1 %	-0.1 %
8		C-Objectivo	4.7 %	-0.6 %
9	↑	R	3.1 %	+0.5 %
10	↑	Swift	3.0 %	+0.4 %



PYPL PopularitY of Programming Language index `https://sites.google.com/site/pydatalog/pypl/PyPL-PopularitY-of-Programming-Language`



PYPL PopularitY of Programming Language index

`http://pypl.github.io/PYPL.html`

http://www.raspberrypi.org/

Se puede comprar en www.newark.com.mx

Python no lo recomendado para cómputo intensivo

- Comparacion de un programa intensivo en punto flotante en tres versiones, python, perl y C.

```
$ time python chuaBinX.py 0.1 1000 12 > secPy

real    3m1.915s
user    3m1.869s
sys 0m0.048s
$ time perl chuaBinX.pl 0.1 1000 12 > secPl

real    3m31.458s
user    3m31.408s
sys 0m0.050s
$ time ./chuaBinX 0.1 1000 12 > secC

real    0m0.982s
user    0m0.968s
sys 0m0.003s
```

Si se ejecutase 1000 veces el trabajo anterior:

```
                        Python  Perl    C
Tiempos de ejecucion     3'2"   3'31"   1"
Veces más                182    211     1
Si fuesen 10^6 bits       ^      ^
   10^6/10^3 = 10^3       |      |      1000" = 16'40"
                          |   un poco más de 58 horas y media
                       50 horas y media
```

Moraleja: No usar lenguajes interpretados en cómputo intensivo

Lenguaje	Mapa caótico	Tiempo (seg)
C	Corrimiento de Bernoulli	0.24
	Mapa casa de campaña	0.40
	Mapa Zigzag	0.48
Python	Corrimiento de Bernoulli	7.68
	Mapa casa de campaña	9.81
	Mapa Zigzag	10.84
Matlab	Corrimiento de Bernoulli	37.73
	Mapa casa de campaña	38.7
	Mapa Zigzag	38.55

Tiempos medidos por Esteban Torres en su trabajo de licenciatura

Una muy breve introducción a Unix

Algunos comandos básicos de Unix

1. `ls` – lista el contenido del directorio
2. `ls -l` – listado largo del contenido del directorio
3. `mkdir nombre` – crea el directorio `nombre`
4. `rmdir nombre` – borra el directorio `nombre`
5. `cd nombre` – se cambia al directorio `nombre`
6. `touch documento` – crea un archivo vacío llamado `documento`
7. `mv documento doc1.txt` – cambia de nombre el archivo `documento` a `doc1.txt`
8. `file doc1.txt` – identifica el archivo `doc1.txt`
9. `rm doc1.txt` – borra el archivo `doc1.txt`

Hay que ser ordenados para trabajar

Hay que crear siempre un espacio de trabajo, por ejemplo:

```
cd
mkdir CursoPython
cd CursoPython
ls -l
nano hola.py
```

Mi primer programa en python

```
print( "¡Hola mundo!" )
```

El programa se ejecuta en la línea de comandos como:

```
$ python hola.py
```

Programando en python

Variables en python

```
x = 123                 # un entero
x = 123L                # un entero largo
x = 3.14                # un real en doble precisión
x = "hola"                  # una cadena
x = [0,1,2]                 # una lista
x = (0,1,2)                 # una tupla
x = open('hello.py', 'r')    # un archivo
```

Sentencia "if"

```
x = 4
if x < 5 :
    print( "x es menor que cinco" )
else :
    print( "x es mayor o igual que cinco" )
```

Sentencia "if"

```
x = 4
if x == 0 :
    print( "x es igual a cero" )
elif x < 5 :
    print( "x es menor que cinco" )
else :
    print( "x es mayor que cinco" )
```

```
# Una línea de comentarios comienza con un '#'
a = 90
print( a )
"""
  Varias líneas
de comentarios
Fraga 24/03/2014
"""
```

Ciclos

- Se puede usar la sentencia 'for',
- o también la sentencia 'while'
- La sentencia 'for' se usa para iterar en listas

```
# Un ejemplo de uso del 'while'
#
i=0
while i<10 :
    print( i )
    i += 1
```

```
#
# El mismo programa anterior pero con
# una modificación en el 'print':
#
i=0
while i<10 :
    print( i, end=' ' )
    i += 1
```

```
# Un ejemplo del uso del 'for'
#
lista = [2,3,5,9]

for i in lista :
    print( i )
```

El comando 'pydoc'

- La ayuda sobre cualquier sentencia se puede obtener con el comando 'pydoc'.
- Dar, por ejemplo, 'pydoc if' en la línea de comandos.

Se pueden definir funciones (1/2):

```
def suma(a, b) :
    return( a+b )

r = suma( 5, 6)
print( "5 + 6 =", r )
```

Se pueden definir funciones (2/2):

```
def f(x) :
    return( x*x )

i=0
while i<20 :
    y = f(i)
    print( i, y )
    i += 1
```

Las funciones matemáticas se encuentran en el *módulo* math:

```
import math

angulo = math.pi/2

print( math.degrees(angulo), math.sin(angulo), \
       math.cos(angulo) )
```

El módulo numpy nos permite definir matrices:

```
import numpy

A = numpy.zeros((3,2))

# ndim es el número de dimensiones
# size es el número total de elementos
# shape   mxn
print( A.ndim, A.size, A.shape )
i=0
while i<3 :
    j=0
    while j<2  :
        print( A[i,j], end='' )
        j += 1
    print( )
    i += 1
```

- Debe definirse desde el principio del tamaño de la matriz, o
- usar sus métodos asociados 'resize', 'reshape', por ejemplo.

```
import numpy as np

a = np.ones((3,3))
print( a )

a.resize( (3,4 ) )
print( a )
```

Guardando datos en un archivo:

```
def f(x) :
    return( x*x )

arch = open( "salida.txt", 'w' )

i=0
while i<20  :
    y = f(i)
    arch.write( "{} {}\n".format(i, y) )
    i = i+1

arch.close()
```

Argumentos en la línea de comandos:

```
import sys

n = len(sys.argv)
print( n )
if n != 3 :
    print( "Args: número1 número2" )
    sys.exit(1)

print( sys.argv[1], sys.argv[2] )
```

Excepciones:

```
import sys

n = len(sys.argv)
if n != 2 :
    print( "Args: nombre_archivo" )
    sys.exit(1)

nombre = sys.argv[1]
try:
    arch = open( nombre, 'r' )

except IOError:
    print( "No pude abrir el archivo", nombre )
    sys.exit(2)

for linea in arch :
    print( linea )
arch.close()
```

Para separar los tokens de una cadena:

```
a = "Esta es una prueba de una línea de texto"

b = a.split()

for palabra in b:
    print( palabra )

c = b[3]
print( "Otro: ", c )

n = len(b)
i=0
while i<n :
    print( b[i] )
    i = i+1
```

Para leer un archivo de texto con comentarios:

```
#
# El módulo de expresiones regulares en python:
import re

# Este es un comentario, en una línea
a = "#  El comienzo de una línea!"

if  re.match( "^\s*#", a ) != None :
    print( a )
```

Separacion del código en varios archivos:

Archivo 'operaciones.py'

```
def suma( a, b ) :
    return( a+b )

def resta( a, b ) :
    return( a-b )

def multiplicacion( a, b ) :
    return( a*b )
```

```
import operaciones
import sys

n = len( sys.argv )
if n != 3 :
    print( "Args: numero1 numero2" )
    sys.exit(1)

try:
    n1 = float( sys.argv[1] )
except ValueError:
    print("El primer argumento no es un número'
    sys.exit(2)

try:
    n2 = float( sys.argv[2] )
except ValueError:
    print("El segundo argumento no es un númer
    sys.exit(3)

r1 = operaciones.suma( n1, n2 )
r2 = operaciones.resta( n1, n2 )
r3 = operaciones.multiplicacion( n1, n2 )
print( r1, r2, r3 )
```

De: `https://docs.python.org/2/tutorial/classes.html#odds-and-ends`

- Cuando se necesita una estructura (“struct´´) como en C, para agrupar y proteger las variables en todo un programa, se puede utilizar en python la definición de una clase vacía.

```
class Cuadrado:
    pass

def procesa( pd, vx, vy ) :
    pd.x = vx
    pd.y = vy

def imprime( pd ) :
    print( pd.x, pd.y )

Datos = Cuadrado()
imprime( Datos )
Datos.x = 40
Datos.y = 20
imprime( Datos )
procesa( Datos, 1, 2 )
imprime( Datos )
```

Objetos

```
class Empleado :
    pass

listaTodos = []

Juan = Empleado()
Juan.nombre = "Juan Pérez"
Juan.salario = 1000
listaTodos.append(Juan)

Maria = Empleado()
Maria.nombre = "María Hernández"
Maria.salario = 1100
listaTodos.append(Maria)

for obj in listaTodos :
    print( type(obj), obj.nombre, obj.salario )

print( "Ok!" )
```

Diferencia entre variables de la clase y del objeto

```
class Cuadrado:
    p=0
    def __init__( self ) :
        self.p = [0.0, 0.0]

    def imprime( self ) :
        print( Cuadrado.p, self.p )

c1 = Cuadrado()
c2 = Cuadrado()

c1.p[0] = 12
c2.p[1] = 24

c1.imprime( )
c2.imprime( )

Cuadrado.p = 221
c1.imprime( )
c2.imprime( )
```

Otro ejemplo con objetos

```
class punto2D :
    def __init__( self, x, y ) :
        self.x = x
        self.y = y

    def __add__( self, other ) :
        return( punto2D( self.x + other.x,  self.y + other.y) )

    def myprint( self ) :
        print( self.x, self.y )

p1 = punto2D( 1.0, 0.0 )
print( type(p1) )
p1.myprint( )

p2 = punto2D( 5.0, 4.0 )
p2.myprint( )

p3 = p1 + p2
print( type(p3) )
p3.myprint( )
```

De: `https://docs.python.org/2/tutorial/classes.html#random-remarks`

- Atributos de datos sobreescriben los atributos de métodos con el mismo nombre, para evitar conflictos accidentales en los nombres, que pueden causar bugs difíciles de encontrar en programas grandes, se debe usar alguna convención en los nombres.
- Un posible convención podría ser nombrar los métodos con la primera letra en mayúscula, y los datos con un prefijo (tal vez '_', el guión bajo).
- Otra convención podría ser usar verbos para los métodos y nombres para datos.

Resolviendo problemas con la computadora

Fases del desarrollo de software:

0. Entender el problema.
1. Análisis del problema y especificaciones.
2. Diseño del sistema.
3. Codificación e integración.
4. Verificación y validación.
5. Mantenimiento del sistema.

Algoritmo

Es un procedimiento que se ejecuta en una computadora, y este procedimiento está compuesto por instrucciones que realizan:

1. Las entradas y salidas del algoritmo.
2. No debe ser ambiguas, de manera que es claro lo que cada instrucción representa cuando se ejecuta.
3. Deben ser simples para que puedan ser llevados a cabo por una computadora.
4. Deben de ser finitas, esto es, el algoritmo debe terminar después de un número finito de operaciones.

Pseudocódigo

Es la representación de un algoritmo usando palabras comunes y álgebra.

Los **algoritmos** se diseñan usando tres estructuras de control básicas:

1. Secuencia: Los pasos son realizados de una manera estrictamente secuencial, cada paso siendo ejecutado exactamente una vez.
2. Selección: Una de varias acciones alternativas es seleccionada y ejecutada.
3. Uno o más pasos se realizan repetidamente.

Estas tres mecanismos de control son muy simples individualmente pero, de hecho, la construcción de cualquier algoritmo con estas estructuras fue demostrado por Bohn y Jacopini en 1966 [1].

[1] C. Bohn and G. Jacopini, "Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules", *Communications of the ACM*, Vol. 9, No. 5, Mayo 1966, pp. 336-371.

Problemas

Problema 1: Resolver una ecuación lineal (1/2)

1. ¿Qué es una ecuación lineal?

Problema 1: Resolver una ecuación lineal (1/2)

1. ¿Qué es una ecuación lineal?
2. ¿Qué significa resolver una ecuación?

Problema 1: Resolver una ecuación lineal (1/2)

1. ¿Qué es una ecuación lineal?
2. ¿Qué significa resolver una ecuación?
3. Si ya entendemos los dos puntos anteriores podemos establecer el pseudocódigo para resolver el problema.

Problema 1: Resolver una ecuación lineal (2/2)

1. La entrada de los valores para las incógnitas será en la línea de comandos
2. La ecuación de la línea: $y = mx + b$
3. La resolvemos $mx + b = 0$, $x = \frac{-b}{m}$
4. Imprimimos el valor de x para el cual y será igual a cero.

La solución:

```
# coding: utf-8

import sys

# Los valores de m y b por la línea de comandos
if len( sys.argv ) != 3 :
    print( "Args: m b" )
    sys.exit(1)

m = float(sys.argv[1])
b = float(sys.argv[2])

x = -b/m
print( x )
```

Problema 2: Solución a una ecuación cuadrática

1. La ecuación cuadrática $ax^2 + bx + c = 0$,

Problema 2: Solución a una ecuación cuadrática

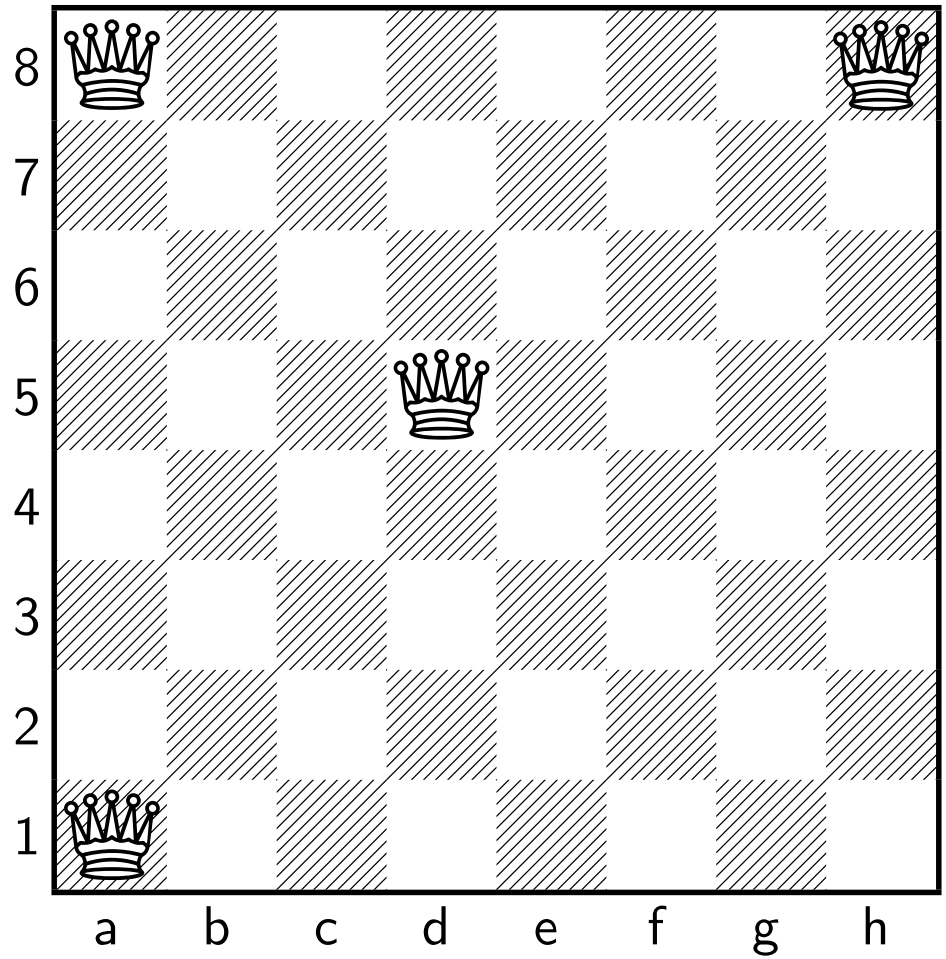
1. La ecuación cuadrática $ax^2 + bx + c = 0$,
2. tiene las soluciones $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Problema 3: Guardar datos en un archivo

Problema 4: Leer datos de un archivo

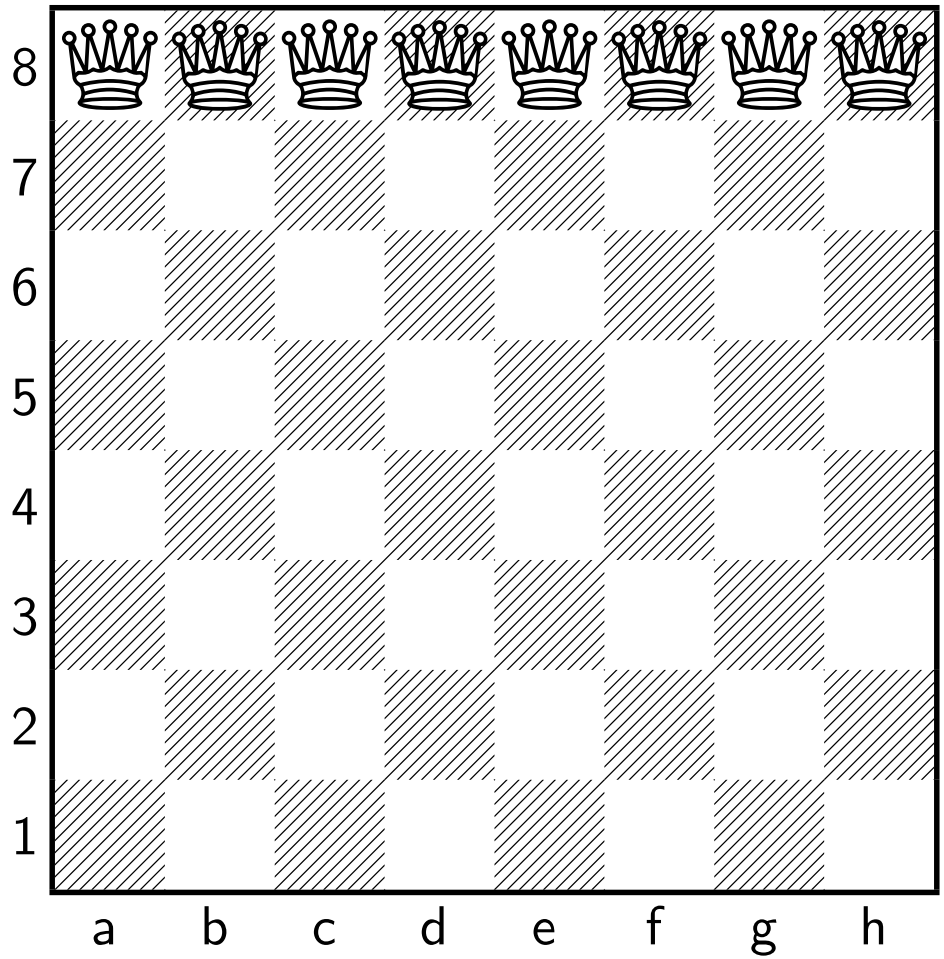
- Se tiene un archivo con datos
- Se tiene que crear un programa que maneje esos datos como una hoja de cálculo

Problema 5: El problema de n damas en el tablero de ajedrez



- Vamos a crear una solución al problema analizando todas las posibilidades
- Esta solución se conoce como “de fuerza bruta”

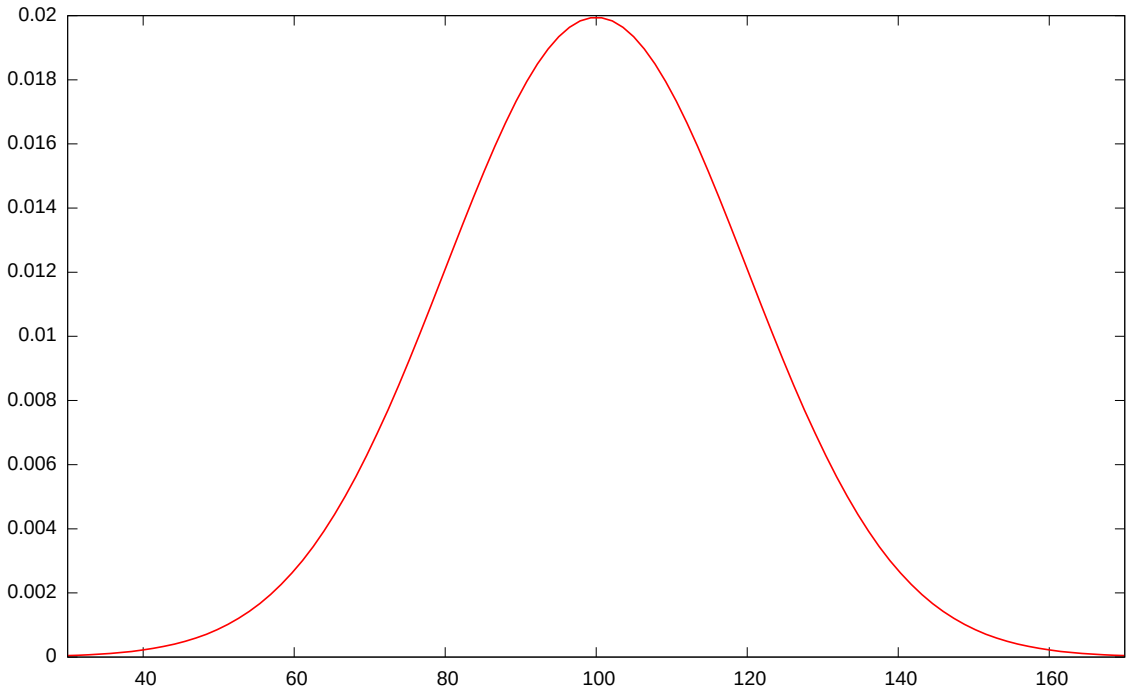
Una posición a evaluar:



1. Para todas las posiciones posibles:
    1.1 Si la posición no presenta un jaque entre damas, se imprime la posición

1. posicion=0 # para contar el número de posiciones válidas
2. Para todas las posiciones posibles:
   - 2.1 Si la posición no presenta un jaque entre damas,
     - 2.1.1 posicion = posicion + 1
     - 2.1.2 se imprime la posición, que es la número 'posicion'
3. El número de posiciones es 'posicion'

Problema 6: Generación de números aleatorios (1/2)

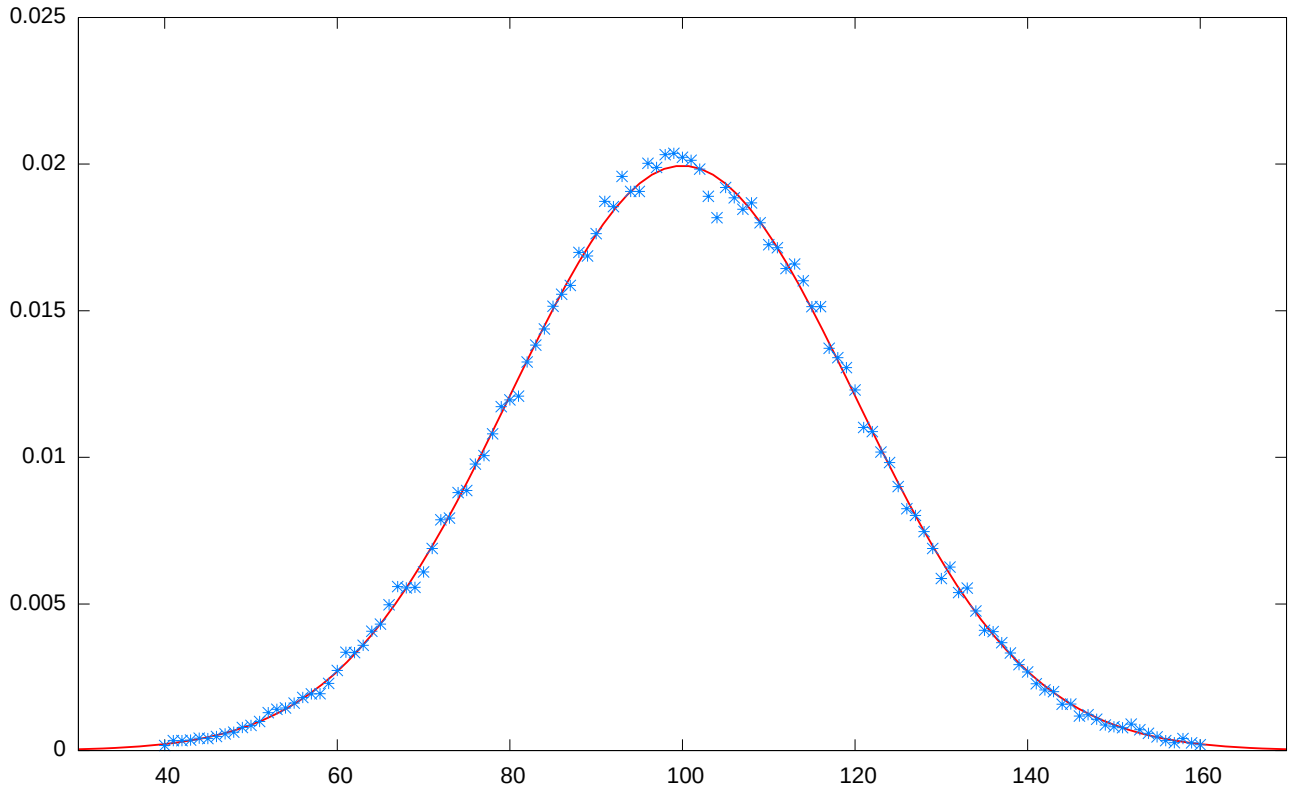


Gráfica de la función $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$,
para $\mu = 100$ y $\sigma = 20$

Problema 6: Generación de números aleatorios (2/2)

```
import random
import sys

if len(sys.argv) != 2 :
    print( "Args: número" )
    sys.exit(1)

veces = int(float(sys.argv[1]))

media = 100
sigma =  20

indice0 = media-3*sigma
indice1 = media+3*sigma

print( "# ", indice0, indice1 )

amplitud = indice1-indice0 + 1

varreglo = [0]*amplitud
i=0
while i<veces :
    n = random.gauss(media, sigma)
    v = int(n+0.5) - indice0

    if v>=0 and v<amplitud :
        varreglo[v] += 1

    i += 1
```

```
i=0
while i<amplitud :
    v = i + indice0
    norma = float(varreglo[i])/veces
    # print( v, varreglo[i] )
    print( v, norma )
    i += 1
```


0.025
0.02
0.015
0.01
0.005
0
40
60
80
100
120
140
160

Problema 7: Animación de gráficas usando gnuplot