Guía del Taller de Programación FQ 2 – 1er Cuatrimestre 2018

Contenidos

1. Introducción a la Programación

Los objetivos de este taller son

- Servir de primera introducción a la computación para aquellos estudiantes que no tienen experiencia en el tema.

- Aplicar programación en Python para estudiar conceptos de la materia FQ2.

El taller tiene un enfoque práctico, es decir, intentaremos que cada alumno pueda escribir sus propios programas, pelearse con Python y avanzar a su propio paso. Por otro lado, no buscaremos resolver problemas específicos de la guía de FQ2, sino escribir programas modelo que permitan resolver tipos o conjuntos de problemas. Por ejemplo, resolveremos un problema de cinética química de una reacción autocatalítica. Con modificaciones menores, es posible modificar dicho programa para otros tipos de problemas. Finalmente, es importante mencionar que el presente taller no es un curso sistemático de programación Python, hay muchas características de este lenguaje que no veremos. Tambien es

importante mencionar que existen otros lenguajes de programación cuya sintaxis difiere en mayor o menor medida de Python (Fortran, C, matlab etc), pero los conceptos básicos de programación a aprender en el curso son similares en todos estos lenguajes y servirán por lo tanto para aprender los mismos más rápidamente.

Existen machismos recursos online sobre Python y cuando no se sabe cómo hacer algo, google es siempre la primera opción. Algunos recursos útiles son:

- http://pythontutor.com/
  Este sitio permite ingresar código Python simple y correrlo línea por línea. Es didáctico para aquellos que recién comienzan y para usuarios más avanzados que quieran encontrar errores de código.
- https://pythonbasics.org/
  Uno (de muchos) sitios que ofrece una introducción al lenguaje Python. Es más amplio que esta guía dado que tiene un foco más general.
- https://s3.amazonaws.com/assets.datacamp.com/blog_assets/PythonForDataScience.pdf
  Una (de muchas) hojas de resumen de Python. Resume la sintaxis del lenguaje en una hoja para su consulta rápida.
- https://repl.it/repls/PowderbluePriceySales
  Este sitio permite correr código de Python online.

Python permite ingresar comandos uno a uno en la línea de comandos o correr programas desde la línea de comandos. El programa de ejemplo, *primos.py*, busca números primos, y luego los imprime, grafica y graba en disco. Para ejecutarlo hay que hacer:

```
python3 primos.py
```

La estructura del programa esta comentada (todo lo que sigue a "#" son comentarios en Python) para ser autoexplicativa. Para poder entenderla, sin embargo, es necesario comprender una serie de conceptos de programación. En el resto de esta sección describiremos brevemente estos conceptos y su implementación en Python, con foco en aquellos comandos que emplearemos más adelante en el taller.

1.1. Variables

Las variables sirven para guardar datos en memoria. Existen distintos tipos de variables, los más usados son:

i) Enteros (integer): guardan números enteros

```
maxnumero = 100
```

ii) Reales/Coma Flotante (float o real): guardan números reales

```
pi = 3.14159
```

iii) Textos (strings): guardan caracteres. Se usan las comillas simples o dobles para delimitar el texto.

```
filename = "primos.txt"
```

iv) Lógicos (Boolean): solo pueden valer *True* o *False*

```
flagprimo = True
```

Además los enteros y reales pueden tener distintos tamaños (por. ej se habla de reales de doble o simple precisión). Python (a diferencia de otros lenguajes) permite declarar el valor de una variable sin declarar su tipo primero.

1.2. Operaciones con variables

Es posible realizar operaciones aritméticas simples con las variables numéricas, por ejemplo:

```
pi = 3.14159
r = 10.0
area = pi*r**2
print (area) # print sirve para imprimir en pantalla!
```

Otro ejemplo, cálculo del resto:

```
numerador = 10
divisor = 3
resto = numerador % divisor
print (resto)
```

Otro ejemplo típico: incrementar en 1 una variable:

```
counter = 0
counter = counter + 1
print (counter)
```

También pueden hacerse operaciones con textos:

```
name = "salida"
extension = ".out"
file = name + extensión
print(file)
```

Un ejemplo medio raro:

```
texto = "#!"
texto = texto*10
print(texto)
```

A veces queremos combinar variables de distinto tipo. Para ello es necesario transformarlas de un tipo a otro, por ejemplo:

```
name = "salida"
index = 15
extension = ".out"
file = name + str(index) + extension
print(file)
```

1.3. Arrays

Los arrays (en Python llamadas listas) son vectores o matrices de datos. Los elementos del array son variables de algunos de los tipos mencionados más arriba (todos los elementos de un array son del mismo tipo!). Los arrays pueden definirse entre [] y la posición dentro del array se indica usando [] después del nombre la variable:

```
lista = [10, 20, 40]
print(lista[0], lista[2])
```

Este último ejemplo imprime "10 40" porque el primer elemento del array en Python tiene índice "0" (en otros lenguajes como Fortran, el primer elemento tiene índice 1). Se puede crear un array vacío haciendo:

```
lista = []
```

y luego agregar elementos al mismo mediante el método append:

```
lista.append(10)
print(lista[0])
```

A veces queremos crear un array vacio de cierto tamaño, por ejemplo:

```
lista=[0.0]*10
```

crea un array de 10 elementos, todos iguales a cero. Finalmente los arrays pueden tener dimensión mayor a uno, por ejemplo

```
lista=[[1, 2], [10, 20]]
print(lista[0][1])
```

1.4. Bloques de control

Los bloques de control permiten dirigir la ejecución del código, creando repeticiones (loops) o bifurcaciones en el mismo.

Bifurcaciones: Las bifurcaciones se crean con el comando `if`. Veamos el ejemplo

```
valor = 3
if (valor == 4):
    print("Es 4")
if (valor > 4):
    print("Es mayor que 4")
    print("bajar el valor!")
print("El valor es:", valor)
```

Notar el que el uso de las indentaciones (espacios al comienzo de las líneas) indica que código se encuentra dentro del bloque `if` y solo se ejecutará si la condición que procede a `if` es cierta. En general el número de espacios empleados no es importante, pero hay que ser consistente (por ejemplo usar siempre 4 espacios, no conviene usar TAB). Otra cosa que hay que notar es que la condición de igual se determina empleando "==" para no confundirse con el "=" empleado en la asignación de variables. Finalmente, al comenzar cada bloque se emplean ":". Olvidarse los ":" es un error muy común.

Loops: Existen distintas formas de crear un loop, las más comunes son empleado los comandos `for` y `while`. En este tutorial describiremos solo el comando `for`.

El comando `for` permite ejecutar el código en el bloque, asignando a una variable dentro del mismo distintos valores. Por ejemplo:

```
valores = [3, 5, 10]
for i in valores:
    print(i)
```

La función `range` es útil combinada con el comando `for`. Requiere dos argumentos (inicio y fin) y puede tomar un tercero (paso). Por ejemplo:

```
for i in range(1,10):
    print(i)
for i in range(10,1,-1):
    print(i)
```

Los bloques pueden ponerse uno dentro del otro (tener cuidado con las indentaciones!!!). Por ejemplo

```
for i in range(1,10):
    if (i == 5):
        print("Cinco")
    print(i)
```

1.5. Funciones

Python permite definir funciones. Las funciones toman argumentos y devuelven un resultado. Por ejemplo

```
def f(x,y):
    return x*y
print(f(3,4))
print(f(1,10))
```

1.6. Escribir a disco

Python puede leer y escribir en archivos en el disco. Vamos a aprender a escribir a disco, lo cual va a ser muy útil para grabar nuestros resultados.

Primero hay que abrir el archivo donde queremos grabar:

```
f = open("test.txt","w")
```

Notar que "w" indica que vamos a grabar en el archivo. La escritura en el archivo se realiza con el método write:

```
f.write("Esto va al archivo \n")
```

El "\n" al final se usa para comenzar una línea nueva.

1.7. Módulos

Los módulos son librerías de funciones. Existen muchas de estas librerías pre-creadas para fines específicos. En particular, las librerías más interesantes para nosotros serán:

- *matplotlib*: permite hacer gráficos fácilmente
- *random*: sirve para generar números al azar
- *numpy*: contiene distintas funciones científicas y de análisis numérico y permite trabajar con arrays (no hace falta en el presente curso, pero puede usarse para cosas más avanzadas)

Ejemplo de graficos:

```
import matplotlib.pyplot as plt
a = [1, 2, 3 , 4,  5, 6 , 7, 8]
s = []
for i in a:
    s.append(i**2)
plt.plot(a,s, "ro")
plt.show()
```

Ejemplo de números al azar:

```
import random
for j in range(1,11):
    print(random.randint(0,10))
    print(random.uniform(0,10))
```

1.8. Programa de Ejemplo

La introducción de las secciones anteriores nos permite entender, correr y modificar el programa *primos.py*. Basados en el mismo, proponemos el siguiente ejercicio:

Ejercicio: Escribir un código Python que encuentre los primeros 20 miembros de la serie de Fibonacci (1, 1, 2, 3, 5, 8, ….), los imprima en orden y en orden inverso y los grafique.

2. Diferencias Finitas: Aplicación en Cinética Química

El método de diferencias finitas reemplaza las derivadas en ecuaciones diferencias en diferencias (lo cual resulta muy intuitivo). Veamos un ejemplo muy simple:

$$A \rightarrow B$$

$$\frac{d[A]}{dt} = -k_1[A] \tag{1}$$

Esta ecuación posee condiciones iniciales $t = 0$ y $[\underline{A}](t = 0) = [A]_0$.

En diferencias finitas, discretizamos el tiempo en pasos Δt:

$$\frac{[A]^{i+1} - [A]^i}{\Delta t} = -k_1[A]^i \tag{2}$$

donde $[A]^i$ indica la concentración de tiempo en el paso i. Por supuesto, si conocemos $[A]^i$, podemos calcular $[A]^{i+1}$:

$$[A]^{i+1} = [A]^i - k_1[A]^i \Delta t \tag{3}$$

Ahora es posible calcular $[A]^{i+2}$, $[A]^{i+3}$, etc.

Ejercicio: Escribir un código Python que calcule la concentración de [*A*] para un mecanismo autocatalitico:

$$A + X \rightarrow 2A$$
$$X \rightarrow P$$

En realidad, en el lado derecho de la ecuación (2) también sería posible emplear $[A]^{i+1}$:

$$\frac{[A]^{i+1} + [A]^i}{\Delta t} = -k_1[A]^{i+1} \tag{4}$$

Despejando $[A]^{i+1}$:

$$[A]^{i+1} = \frac{[A]^i}{(1 + k_1 \Delta t)} \tag{5}$$

Esta forma de resolver el problema se conoce como método implícito (la ecuación (2) se conoce como método explicito). En general los métodos implícitos se emplean cuando surges problemas de inestabilidad numérica, por ejemplo en problemas de reaccion-difusion.

3. Diferencias Finitas: Aplicación en Procesos de Transporte

Consideremos la segunda ley de Fick en una dimensión:

$$\frac{dc}{dt} = D\frac{d^2c}{dx^2} \tag{6}$$

Podemos discretizar esta ecuación tanto en el espacio (paso Δx) como en el tiempo (paso Δt):

$$\frac{c^{i+1}(j) - c^i(j)}{\Delta t} = D\frac{\frac{dc^i}{dx}(j+1) - \frac{dc^i}{dx}(j)}{\Delta x} = D\frac{c^i(j+1) - 2c^i(j) + c^i(j-1)}{\Delta x^2} \tag{7}$$

En esta notación c^i(j) es la concentración en el paso de tiempo i en la celda j. Es conveniente dividir esta ecuación por una concentracion caracteristica c^0 y reordenar. Finalmente se llega a:

$$\bar{c}^{i+1}(j) - \bar{c}^i(j) = \bar{D}\left[\bar{c}^i(j+1) - 2\bar{c}^i(j) + \bar{c}^i(j-1)\right] \tag{8}$$

Donde $\bar{c} = c/c^0$ y $\bar{D} = D\Delta t/\Delta x^2$. Notar que $\bar{c}$ y $\bar{D}$ no tienen unidades (son adimensionales), lo cual resulta útil porque: 1) no hay que preocuparse de las unidades mientras programamos (hay que ocuparse al proporcionar los parámetros y analizar la salida!), 2) se aprecia mejor la física del problema y se evitan cálculos repetidos (por ejemplo, la ecuación (8) no depende del valor absoluto de la concentración inicial).

A partir de la ecuación (8) podemos despejar las concentraciones en le paso i+1 si conocemos aquellas en i:

$$\bar{c}^{i+1}(j) = \bar{D}\left[\bar{c}^i(j+1) - 2\bar{c}^i(j) + \bar{c}^i(j-1)\right] + \bar{c}^i(j) \tag{9}$$

Hay que prestar mucha atención a los bordes de la caja, es decir valor mínimo de j (j = 0) y máximo (j = N). Si queremos evaluar la ecuación anterior en j = 0:

$$\bar{c}^{i+1}(0) = \bar{D}\left[\bar{c}^i(1) - 2\bar{c}^i(0) + \bar{c}^i(-1)\right] + \bar{c}^i(0) \tag{10}$$

encontramos que *c*(-1) esta fuera de la caja!!!. Tanto en la ecuación diferencial como en su versión discretizada necesitamos una condición de contorno. La más simple es la de concentración constante:

$$\overline{c}^{i+1} = \overline{c}^{b} \tag{11}$$

donde $\overline{c}^{b}$ es una concentración independiente del tiempo. La otra alternativa es fijar el flujo en el borde.

Además de las condiciones de borde en la dimensión x, es necesario fijar condiciones iniciales en el cálculo.

Ejercicio: Escribir un código Python que calcule el proceso de difusión. Para ello, comience con una función tipo pulso:
$\overline{C}$ (x) = 1.0 si |x-L/2|< *a*/2, $\overline{C}$ (x) = 0 en caso contrario (donde *L* es el tamaño de la caja de calculo).
Elegir Δx = *a*, de forma tal que una celda tendrá concentración 1.0 y el resto concentración 0.
Emplear L/Δx = 1000, $\overline{D}$ = 0.1 y condiciones de borde $\overline{C}$ (0) = $\overline{C}$ (*L*) = 0.
Graficar la evolución de la concentración cada 200 pasos de tiempo.

Ejercicio: Modificar el programa anterior para que comience con una función tipo escalón:
$\overline{C}$ (x) = 1.0 si x < *L*/2, $\overline{C}$ (x) = 0 si *x* > *L*/2.
Emplear condiciones de borde $\overline{C}$ (0) = 1.0 y $\overline{C}$ (*L*) = 0.

4. Camino al Azar

Consideremos un camino al azar en una dimensión: imaginemos N cajas y una partícula. La partícula inicialmente se encuentra en la caja j (supongamos j = N/2) y puede saltar a la caja j-1 o j+1 con idéntica probabilidad (*p* = 0.5). Se realizan M saltos y se guarda la posición final de la partícula. Se repite el proceso para N_p partículas más. El resultado de este proceso puede representarse como un histograma (número de partículas que reside en la caja j al final de la simulación en función de j).

Ejercicio: Escribir un código Python que realice el proceso de camino al azar descripto más arriba. Realice el cálculo para valores de M entre 100 y 1000 en incrementos de 100 y grafique todos los histogramas en un mismo gráfico.

También es posible visualizar la trayectoria de partículas individuales. Para ello debemos grabar un archivo xyz que contenga las posiciones de las mismas. Supongamos que hay N_p partículas y M pasos de tiempo. El archivo debe contener M bloques consecutivos (uno para cada paso de tiempo). Cada bloque comienza con una línea que contiene el número de partículas, luego otra que contiene el paso de tiempo y luego N_p líneas que contienen las coordenadas de las partículas. Estas últimas líneas pueden escribirse con el siguiente código:

```
salida.write(" C  "+str(xpos[j]) + "     "+str(ypos[j])+"     0.0     1.0
\n  ")
```

Donde salida es el archivo de salida abierto como `open`, `xpos[j]` y `ypos[j]` son las coordenadas x e y de la partícula j.

Ejercicio: Escribir un código Python que realice el proceso de camino al azar de 1000 partículas en dos dimensiones, guardar las trayectorias en el formato xyz descrito arriba y visualizarlas con el programa VMD.