# Introduccion a la Programacion en Python

El programa primos.py es un ejemplo sencillo formulado para comprender una serie de conceptos de programación y sintaxis de Python. Existen otros lenguajes de programación cuya sintaxis difiere en mayor o menor medida con Python (Fortran, C, etc), pero los conceptos básicos de programación a aprender en el curso servirán para aprender estos lenguajes mas rápidamente.

Python permite ingresar comandos uno a uno en la línea de comandos o correr programas desde la línea de comandos. Alternativamente, la página <http://pythontutor.com/> es excelente para probar programas simples y ver su ejecución en tiempo real.

El programa de ejemplo, primos.py, busca números primos, y luego los imprime, grafica y graba en disco. Para ejecutarlo:

python3 primos.py

La estructura del programa esta comentada (todo lo que sigue a “#” son comentarios en Python) para ser autoexplicativa. Para poder entenderlo, sin embargo, es necesario comprender una serie de conceptos de programación.

Existen muchismos recursos online sobre Python y cuando no se sabe como hacer algo, buscar en google es la primera opción. Algunos recursos útiles:

* <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 mas avanzados que quieran encontrar errores de código.

* <https://pythonbasics.org/>

Uno (de muchos) sitios que ofrece una introducción al lenguaje Python. Es mas amplio que esta guía dado que tiene un foco mas 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.

En el resto de esta sección decribiremos brevemente la sintaxis Python, con foco en aquellos comandos que emplearemos mas adelante en el taller.

## Variables

Las variables sirven para guardar datos en memoria.

Existen distintos tipos de variables. Los mas usados son:

* Enteros (integer): guardan números enteros

maxnumero = 100

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

pi = 3.14159

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

filename = "primos.txt"

* Logicos (Boolean): solo pueden valer *True* o *False*

flagprimo = True

Ademas los enteros y reales pueden tener distintos tamaños (por. ej se habla de reales de doble o simple presicion).

Python (a diferencia de otros lenguajes) permite declarar el valor de una variable sin declarar su tipo primero.

## 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, calculo 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)

Tambien 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, por ejemplo:

name = "salida"

index = 15

extension = ".out"

file = name + str(index) + extension

print(file)

## 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 mas 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 [] despues del nombre la variable:

lista = [10, 20, 40]

print(lista[0], lista[2])

Este ultimo ejemplo imprime “10 40” porque el primer elemento del array en Python tiene indece “0” (en otros lenguajes como Fortran, el primer elemento tiene índice 1). Se puede crear un array vacio 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])

## Bloques de control

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

### **Bifurciones**

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 identaciones (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 numero 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 mas 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 dintintos valores. Por ejemplo:

valores = [3, 5, 10]

for i in valores:

print(i)

La funcion 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 identaciones!!!). Por ejemplo

for i in range(1,10):

if (i == 5):

print("Cinco")

print(i)

## 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))

## 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.

## Modulos

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

* matplotlib: permite hacer graficos 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 mas avanzadas)

Ejemplo:

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 numeros al azar:

import random

for j in range(1,11):

print(random.randint(0,10))

print(random.uniform(0,10))

## Programa de Ejemplo

La introduccion 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. |

# Diferencias Finitas: Apliacion en Cinetica Quimica

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





Esta ecuación posee condiciones iniciales *t* = 0 y [*A*](*t* = 0) = [*A*]0.

En diferencias finitas, discretizamos el tiempo en pasos Δt:



donde [A](i) indica la concentracion de tiempo en el paso i. Por supuesto, si conocemos [A](i), podemos calcular [A](i+1):



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: |

En realidad, en el lado derecho de la ecuación también seria posible emplear [A](i+1):



Despejando [A](i+1):



Esta forma de resolver el problema se conoce como método implícito (la ecuación 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.

# Diferencias Finitas: Aplicacion en Procesos de Transporte

Consideremos la segunda ley de Fick en una dimensión:



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



En esta notación ci(j) es la concentracion en el paso de tiempo i en la celda j. Es conveniente dividir esta ecuación por una concentracion caracteristica *c*0 y reagrupar:



Donde = *c*/*c*0 y ** = *D*Δ*t*/Δ*x*2. Notar que y ** 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 no depende del valor absoluto de la concentracion inicial).

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



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



Pero c(-1) esta fuera de la caja!!!. Tanto en la ecuación diferencial como en su versión discretizada necesitamos una condición de contorno. La mas simple es la de concentracion constante:



donde es una concentracion independiente del tiempo. La otra alternativa es fijar el flujo en el borde.

Ademas de las condiciones de borde en la dimensión x, es necesario fijar condiciones iniciales en el calculo.

|  |
| --- |
| **Ejercicio**: Escribir un código Python que calcule el proceso de difusión. Para ello, comience con una función tipo pulso:  (x) = 1.0 si |x-L/2|< *a*/2, (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,= 0.1 y condiciones de borde (0) = (*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 escalon:  (x) = 1.0 si x < *L*/2, (x) = 0 si *x* > *L*/2.  Emplear condiciones de borde (0) = 1.0 y (*L*) = 0. |

# Camino al Azar