# MATEMÁTICA Y PROGRAMACIÓN

Sylvia da Rosa

Daniela Pagés

Franco Vuan

Luis Langon

María Teresita Carrión

Patricia Añón

Santiago Martorell

Santiago Vigo

Teresa Pérez

# ÍNDICE

# ACORTANDO DISTANCIAS

## Informe de la Pasantía

* Algoritmia e integración de programación al proceso de resolución de problemas en cursos de matemática e informática de la enseñanza media.

Pedeciba Área Informática, investigadora Sylvia da Rosa,
Instituto de Computación - Facultad de Ingeniería

## Introducción

En esta pasantía se han elaborado algunas secuencias didácticas y actividades para el aula, dirigidas a los profesores, con las que se busca que los estudiantes:

- Reconozcan un problema algorítmico, puedan enunciarlo, busquen una solución y la expresen como un algoritmo.
- Aprendan los fundamentos básicos de un lenguaje de programación (en este caso python), y lo utilicen en la implementación de los algoritmos que han escrito en lenguaje natural y/o matemático.
- Refuercen, a través de las actividades anteriores, los conocimientos matemáticos que utilizan en la elaboración de los algoritmos.

En este material se incluyen los problemas seleccionados y trabajados por los pasantes, como una primera aproximación. Complementa el material, un breve manual de python que recoge fundamentalmente los aspectos que surgieron durante este trabajo.

## Objetivos

*El objetivo principal* de este trabajo consiste en fomentar la interacción entre docentes e investigadores del área informática y profesores de la enseñanza media (de matemática y/o de informática), para introducir aspectos de la informática como ciencia básica en dicho nivel del sistema educativo.

*El objetivo específico* de este trabajo consiste en introducir aspectos del pensamiento algorítmico en cursos de matemática (y/o de informática), tomando la definición de pensamiento algorítmico dada por Jeannette Wing [4]. Uno de los aspectos más importantes que señala la autora, es que el factor principal del aprendizaje de informática es la conceptualización del proceso de solución de un problema, y no el uso de las herramientas tecnológicas de que disponemos (que constituyen sin duda una ayuda importante). Los problemas que tratamos son algorítmicos, por lo cual una solución puede expresarse mediante un algoritmo. Otro aspecto importante es que integramos una etapa de implementación de la solución construida en un lenguaje de programación (python (http://www.python.org/) es decir construimos un programa. De esta forma es posible introducir el aprendizaje de conceptos básicos de programación en la enseñanza media, que es parte del objetivo específico.

## Metodología

Para la elaboración de las distintas secuencias y/o actividades, el equipo se subdividió en pequeños grupos, que trabajaron con distintas temáticas del curriculum de Ciclo Básico y Bachillerato de Educación Secundaria y Técnico Profesional.

La metodología empleada apunta a la conceptualización del proceso de resolución de problemas algorítmicos y al aprendizaje de sus distintas etapas.

Como introducción a dicho aprendizaje se abordan problemas matemáticos sencillos, seleccionados por los profesores pasantes de los siguientes temas:

- Divisibilidad
- Geometría analítica
- Determinación del dígito verificador de la cédula de identidad
- Operaciones con conjuntos
- Conectivos lógicos
- Sistema posicional de numeración

El proceso de resolución se divide en la especificación del problema (algorítmico), el diseño de una solución, y la implementación en un lenguaje de programación (python). Un problema algorítmico [1] consiste en:

- Una especificación de una colección válida, posiblemente infinita de conjuntos de entrada,
- Una especificación de los elementos de salida deseados en función de los de entrada.

El término especificación representa una descripción sin ambigüedades. ¿Cómo plantear el problema? Este es uno de los primeros aspectos a tener en cuenta en la introducción de conceptos de algoritmia. Muchos de los errores de los estudiantes provienen de la no comprensión de la especificación del problema planteado, muchas veces porque dicha especificación es ambigua y/o incompleta. Esto los lleva a confundir el problema (¿qué se quiere obtener?) con su solución (¿cómo se debe proceder?)

Para diseñar un algoritmo que solucione el problema se usa, en este trabajo, el lenguaje natural (español) en primera instancia. De este modo los estudiantes conceptualizan los pasos necesarios para arribar a una solución, sin preocuparse de detalles del lenguaje de programación. Una vez trabajado el algoritmo en lenguaje natural, se introduce la etapa de implementación en el lenguaje python, conceptualizando el pasaje de un lenguaje al otro. La interacción entre una y otra etapa es dialéctica, por ejemplo, al implementar se pueden detectar fallas en el diseño.

## Desarrollo

Algunos aspectos teóricos estudiados y discutidos en la pasantía fueron:

* El concepto de informática

Acordamos con Gilles Dowek [3] en utilizar el término "informática" en el sentido de que

los estudiantes aprendan a escribir un programa. Según este autor, el aprendizaje de un lenguaje de programación y de algoritmos elementales, puede aportar mucho a su desarrollo personal y a su relación con el conocimiento. El mismo autor agrega como beneficio del escribir programas, que el estudiante aplica y pone en práctica sus conocimientos. También, que la elaboración de programas tiende un puente entre el lenguaje y la acción, ya que un programa tiene la doble cualidad de ser un texto, y a la vez ser ejecutable. Y finalmente, Dowek considera que el aprendizaje de la informática permite al estudiante aprender el rigor de la ciencia, no por la sanción del docente, si no porque la falta de rigor impide que los programas funcionen. Y esto tiene el valor agregado de permitir mejorar el aprendizaje, por parte de los estudiantes de, por ejemplo, el uso correcto de los cuantificadores.

### * El pensamiento computacional

Pensamos que incluir la elaboración de algoritmos en la educación media ayudará al desarrollo del pensamiento computacional, que "involucra la resolución de problemas, el diseño de sistemas y la comprensión del comportamiento humano, basados en los conceptos fundamentales de la ciencia de la computación" como plantea J.Wing en [4]. Además plantea que al resolver un problema deberían surgir como interrogantes: ¿Cuán difícil es hallar una solución? ¿Cuál es el mejor camino para resolverlo? ¿La solución que encontré es suficientemente buena? El pensamiento computacional nos permite transformar un problema difícil en otro que sepamos resolver. También incluye, según Wing, el uso de la abstracción y descomposición al enfrentarse a una tarea compleja. Significa elegir las representaciones adecuadas y/o modelar los aspectos relevantes de un problema para hacerlo más manejable. El pensamiento computacional consiste en usar la heurística al razonar buscando una solución, planificar, esquematizar y aprender frente a la incertidumbre.

### * Los estándares curriculares del NCTM

Entre los estándares curriculares planteados por el National Council of Teachers of Mathematics [6], en la parte dedicada a los estándares de proceso, se plantea:

### * Resolución de problemas

"... los estudiantes debieran ser estimulados a reflexionar sobre sus razonamientos durante el proceso de resolución de problemas, de manera tal que sean capaces de aplicar y adaptar las estrategias que han desarrollado en otros problemas y contextos." Nos parece que el planteo de problemas algorítmicos y la consecuente elaboración de algoritmos por parte de los estudiantes, colabora con este estándar.

### * Comunicación

"La comunicación matemática es un camino para compartir y clarificar ideas matemáticas. A través de la comunicación, las ideas se transforman en objetos de reflexión, perfeccionamiento, discusión y rectificación. Cuando se motiva a los estudiantes a comunicarse con otros estudiantes sus resultados y razonamientos, sea en forma oral o escrita, ellos aprenden a ser más claros, convincentes y precisos en el uso del lenguaje matemático. Las explicaciones dadas por los estudiantes deben incluir argumentos matemáticos y racionales, no solamente descripciones de procedimientos y resúmenes. A su vez, escuchando las explicaciones de otros, los estudiantes podrán desarrollar sus propias comprensiones." Pensa-

mos que en la redacción de los algoritmos, y la explicación de los fundamentos de la sintaxis elegida, los estudiantes se verán enfrentados a la comunicación constante, ya sea con sus compañeros como con el docente.

* Representaciones

" Las ideas matemáticas pueden ser representadas en formas variadas: imágenes, materiales concretos, tablas, gráficos, números y letras, hojas de cálculo, y muchas otras más. Las formas en las cuales se representan las ideas matemáticas son fundamentales para determinar cómo las personas comprenden y utilizan esas ideas. ... Cuando los estudiantes tienen acceso a las representaciones matemáticas y a las ideas que éstas expresan, y cuando además los estudiantes pueden crear representaciones para capturar conceptos matemáticos o relaciones, ellos adquieren un conjunto de herramientas que expanden significativamente su capacidad para modelar e interpretar fenómenos físicos, sociales y matemáticos." La escritura de un algoritmo constituye una representación escrita en lenguaje natural, y su implementación constituye otro tipo de representación, más vinculada con la representación simbólica que usa la matemática. Esta conversión de registros le da fuerza conceptual a la actividad.

* Los estándares de la CSTA

Por otro lado, cabe señalar que la introducción de la informática en la Enseñanza Media como ciencia básica es preocupación de comunidades académicas internacionales [5,2] y nacionales, que intentan seguir e implementar recomendaciones elaboradas en los estándares de la CSTA (Computer Science Teacher Association). En particular, es una recomendación fuerte de los referentes en la materia a nivel nacional: la Universidad de la República y el Instituto de Computación de la Facultad de Ingeniería, y el PEDECIBA, a través de, por ejemplo, esta pasantía.

CSTA es una organización cuyo objetivo es dar soporte a la educación en computación tanto en la educación primaria como secundaria y universitaria. En especial, en lo relativo al currículo, trabaja en pos de establecer el currículo descrito en el documento [2] , sobre la educación en ciencia de la computación de los 4-5 años a los 16-19. Esta currícula se basa en el principio de que los estudiantes necesitan la comprensión del lugar de la ciencia de la computación en el mundo moderno y que la disciplina ciencia de la computación comprende principios y habilidades, no provistos por otras asignaturas.

Para el nivel 2 (Educación Secundaria), los estándares plantean como objetivos, entre otros, que el estudiante sea capaz de:

1. Usar los pasos básicos de resolución algorítmica de problemas para diseñar soluciones (ej: enunciado del problema y exploración, examen de casos, diseño, implementación de una solución, testeo, evaluación).
2. Definir un algoritmo como una secuencia de instrucciones que pueden ser procesadas por un computador.
3. Evaluar los modos en que diferentes algoritmos pueden ser usados para resolver el mismo problema.
4. Describir y analizar una secuencia de instrucciones a ser seguida.
5. Evaluar qué tipos de problemas pueden ser resueltos usando modelización y simulación.
6. Usar la abstracción para descomponer un problema en sub problemas.

7. Examinar conexiones entre elementos de la matemática y la ciencia de la computación, incluyendo números binarios, lógica, conjuntos y funciones.

Los cursos de matemática proveen un gran número de problemas interesantes que permiten trabajar para conseguir dichos objetivos.

## Aportes de la investigación

Como se señala en los objetivos, se busca introducir aspectos de la informática como ciencia básica en la enseñanza media, utilizando la riqueza de problemas y situaciones adecuadas que proveen para ello los cursos de matemática. Es necesario para ello, distinguir claramente dicha ciencia de los servicios y productos tecnológicos derivados de su desarrollo.

A propósito de las relaciones entre la tecnología y la educación matemática, citamos a Artigue, M [7]:

"Ciertamente estas tecnologías son social y científicamente legítimas, pero en el nivel de la escuela, esas legitimidades no son suficientes para asegurar su integración, pues no se busca que la enseñanza forme alumnos aptos para funcionar matemáticamente con esas herramientas —lo que sería el caso, por ejemplo, de una formación de carácter profesional— se busca mucho más. Efectivamente, lo que se espera en esencia de esas herramientas es que permitan aprender más rápidamente, mejor, de manera más motivante, una matemática cuyos valores son pensados independientemente de esas herramientas."

" ... Partiendo del principio de que una técnica posee un valor pragmático que corresponde a las potencialidades que ofrece para producir resultados, y también un valor epistémico, en el sentido de que nos ayuda a comprender los objetos que pone en juego. Nos interesan las modificaciones introducidas en el vínculo entre esos dos valores debido a la introducción de herramientas informáticas tales como calculadoras y programas... La investigación debe entonces encontrar los medios de reforzar, mediante la elección de situaciones adecuadas, el valor epistémico de las técnicas instrumentadas, si desea contribuir a asegurar su legitimidad dentro de las instituciones educativas."

Las nuevas tecnologías han dado a la educación herramientas innovadoras, capaces de producir fuertes modificaciones prácticas y operativas en la práctica docente. Sin embargo, tales tecnologías no pueden por sí mismas reemplazar a la pedagogía si no que deben subordinarse a su servicio. El concepto mismo de tecnología educativa refiere básicamente al manejo de instrumental técnico con el propósito de mejorar el proceso didáctico.

La pedagogía y la tecnología siempre han estado relacionadas. El auge que han tenido (y tienen) las tecnologías digitales en la sociedad en los últimos años, ha hecho esa relación mucho más explícita debido a la abundancia de productos y servicios tecnológicos en todas las actividades de la sociedad y especialmente en el sistema educativo. Esa abundancia no ha sido acompañada con la articulación e integración de los recursos e instrumentos de la tecnología en la actividad educativa. El riesgo consiste en que el uso instrumental e irreflexivo de tecnología desplace a la pedagogía de su rol conductor de los procesos de enseñanza-aprendizaje. Disminuir (o evitar) ese riesgo implica una responsabilidad y un compromiso del sistema educativo y es en ese sentido que se ubica esta pasantía.

## Reflexiones finales

Esta pasantía nos ha aportado, en primer lugar, la experiencia del trabajo en conjunto, desde la diversidad de miradas que supone la integración del equipo: profesores de informática, profesores de matemática e investigadores en informática. Fue un desafío para los docentes, en primer lugar aprender el uso del lenguaje python. Esto nos permitió, sin embargo, pasar por las dificultades que tendrán los estudiantes cuando trabajemos con ellos. También tuvimos oportunidad de discutir distintos abordajes posibles de cada uno de los temas que seleccionamos, aportando cada pasante desde su subjetividad.

No fue posible avanzar sobre el uso de Python en GeoGebra, que era una de las motivaciones de la elección de este lenguaje (en la versión 5 de GeoGebra que se está desarrollando, hay una ventana Python). Las razones fueron, por un lado, la falta de tiempo de la pasantía, y por otro, que la versión del lenguaje que aparece en GeoGebra es Jython, que es Python con Java, y esto implica cambios en la forma de usar el intérprete. Nos parece muy importante avanzar sobre esto en el futuro, ya que GeoGebra es un software muy utilizado por los docentes, y el aprendizaje del uso de esta ventana será de provecho para profesores y estudiantes.

### Bibliografía

1. D. Harel, Algorithmics: The Spirit of Computing, Addison-Wesley, Reading, MA, (425 pp.) 1987. 2nd edition, 1992; 3rd edition, 2004.

2. CSTA (Computer Science Teachers Association) http://csta.acm.org/Curriculum/sub/ACMK12CSModel.html, A.T.F.C. Comitee, A Model Curriculum for K-12 Computer Science, Tech. rep., Computer Science Teachers Association (csta), Association for Computing Machinery (ACM), 2003.

3. Gilles Dowek, Quelle informatique enseigner au lycee?, https://who.rocq.inria.fr/Gilles.Dowek/lycee.html (2005)

4. Jeannette M. Wing, Computational Thinking, Communications of the ACM, Vol. 49, Nr. 3, 2006.

5. EPI (Enseignement Public & Informatique ) Association Enseignement Public & Informatique (EPI) (2011), http://www.epi.asso.fr/.

6. NCTM. PRINCIPIOS Y ESTÁNDARES PARA LA EDUCACIÓN MATEMÁTICA. Traducción de Manuel Fernández SAEM Thales Sevilla 2003 ISBN 84-933040-3-4

7. Artigue, M (2004).- Problemas y desafíos en educación matemática: ¿Qué nos ofrece hoy la didáctica de la matemática para afrontarlos?.- Educación Matemática, Diciembre, año/vol. 16, número 003. Santillana. Distrito Federal, México pp 5-28.

# NUMERACIÓN POSICIONAL

Santiago Vigo (Liceo "Tomás Berreta" N°1 Ciudad de Canelones)

## Objetivo

El objetivo de la secuencia es, estudiar la numeración posicional, a través de la elaboración de algoritmos que permitan escribir, en base 10, números expresados en otra base y viceversa.

Por más que la noción de numeración posicional se trabaja desde los primeros años de educación primaria, la expresión en una base distinta de 10 es en general desconocida y su abordaje puede ser una vía de entrada al estudio de algunas propiedades de los números naturales.

Estas actividades pueden ser trabajadas en varios niveles, en 2°DC en matemática específica está concretamente el tema bases de numeración, pero como se mencionó anteriormente se puede trabajar en 1°CB[1] donde se tratan propiedades de los números naturales y operaciones sobre ellos, concretamente división entera.

La algoritmia, que también es un objetivo de estas actividades, es un tema al que los profesores de matemática en general rehuimos, sin embargo los razonamientos necesarios para lograr desarrollar, expresar y evaluar un algoritmo tienen una gran riqueza, en especial en lógica.

Las actividades que se detallan a continuación son ejemplos de cómo se puede abordar el tema, no pretenden ser un bloque de actividades rígido, si no que el profesor puede creer conveniente agregar, intercalar, modificar y/o eliminar algunas.

## Actividad 1

Es una actividad introductoria que pretende provocar la reflexión sobre los temas a tratar y sentar las bases para un trabajo posterior.

Analizaremos cómo escribimos los números:

Cuanto más grande es un número, ¿mayor cantidad de cifras tiene?

Escribe el 88 en números romanos, y escribe el 90.

Discute sobre por qué crees que los números romanos no se usan cotidianamente como los arábigos.

- ¿Qué ventajas tienen los números arábigos sobre los romanos?
- ¿Cuánto vale el 1 en 3124?
- ¿Cuánto vale el 1 en 3412?
- ¿Por qué valen distinto?

Escribe al 3412 en función de potencias de 10

1 2° DC refiere a 2° de bachillerato diversificación científica en enseñanza secundaria, y 1°CB refiere a 1° año de Ciclo Básico el primer año de educación secundaria

## Actividad 2

Introduce la numeración en base 3 a partir de un juego, se toma como disparador para el trabajo posterior.

### Juego

* Preparación:

- Se hace pasar a 3 estudiantes al frente de la clase (pueden ser más).
- Se los ubica mirando de frente a sus compañeros en una hilera.

* Desarrollo:

Cada vez que el profesor aplaude el primer estudiante hará un movimiento, con el primer aplauso, levanta una mano, con el segundo aplauso, manteniendo la mano levantada, levanta la otra mano, con el tercer aplauso el estudiante baja ambas manos, con el cuarto aplauso, levanta una mano otra vez, el ciclo se repite hasta que termine el juego.

El segundo estudiante levantará una mano cuando el primero las baje, levantará la segunda mano (manteniendo la primera levantada) cuando el primer estudiante vuelva a bajar sus manos. Es decir que sube una mano con el tercer aplauso, la otra mano con el sexto aplauso, baja las manos con el noveno, y así.

El tercer estudiante realizará los mismos movimientos y cada vez que el segundo estudiante baja los brazos. Si hay más estudiantes el proceso continúa de la misma manera.

* Objetivo del juego.

Dar la cantidad de aplausos que el profesor realiza cuando, súbitamente, se detiene y escribe en el pizarrón la posición en que quedaron cada uno de los estudiantes.

Nota: Obviamente este objetivo no debe decirse antes de comenzar el juego para evitar que cuenten.

* Preguntas:

¿Se puede saber la cantidad de aplausos que hubo sólo con mirar el pizarrón?
¿Cuántos aplausos hubo?
¿Puedes encontrar algún vínculo con la actividad anterior?
¿Cada cuántos aplausos realiza el segundo estudiante un movimiento?
¿Cada cuántos aplausos realiza el tercer estudiante un movimiento?

Escribe una secuencia de pasos para a partir del dibujo del pizarrón saber la cantidad de aplausos.

## Actividad 3

En esta actividad se formaliza una forma de escribir en base 3 y se genera el algoritmo para pasar de base 3 a base 10, aunque aún se sigue trabajando a partir del juego de la Actividad 2.

Supone que a la posición de los estudiantes los representamos con tres números, cada uno de ellos representando el número de brazos levantados de cada estudiante, por ejemplo, el número [2,1,0] significa que el estudiante de la derecha no tiene los brazos levantados, el del medio un brazo levantado, y el de la izquierda ambos brazos levantados.

¿Cómo hay que proceder para, a partir de una terna tipo [2,1,1], decodificar la cantidad de aplausos que se dieron? Escríbelo esquemáticamente

Utiliza el esquema anterior para saber la cantidad de aplausos que hubo para que los estudiantes queden en las posiciones siguientes:
a) [2,0,0]
b) [1,0,1]
c) [1,1,1]

Escribe la posición para cada uno de los estudiantes para:
a) 20 aplausos
b) 24 aplausos
c) 45 aplausos
d) 80 aplausos

Escribe una secuencia de pasos para que, a partir de un número, se devuelva una lista con la posición en que quedan los estudiantes si se hacen tantos aplausos como indica el número. Escríbelo esquemáticamente.

* Posibles pasos a seguir

La generalización es una parte importante de la actividad matemática de estudiantes a nivel secundario,, aquí presentamos algunas posibles actividades en este sentido.

¿Cuál es el mayor número que podemos representar? ¿Y si ponemos un estudiante más, o 2 o 3...?
Los números que representa cada estudiante nunca son mayores a 2 por un problema antropomórfico, pero ¿qué pasaría si octópodos ocuparan el lugar de los estudiantes?¿Cómo representarían 33 aplausos?
Otra forma de cambiar la base podría ser que los estudiantes representen su cantidad con los dedos de una mano mientras esconden el resto de sus cuerpos atrás del escritorio (base 6) , o representar las cifras parándose y agachándose (base 2).

## Implementación

El profesor tomará los algoritmos hechos por los estudiantes y los ayudará a implementarlos en python.
Los errores en los mismos pueden ser tratados antes de la implementación o después.

## Soluciones posibles para los esquemas del algoritmo.

* Resolución esquemática del problema de pasar de la posición de los estudiantes al número que representa:

El primer número vale tanto como el mismo, así que no hay que hacer nada con él, el segundo número cuenta de a 3, se mueve una vez por cada 3 aplausos por lo tanto hay que multiplicarlo por 3, el tercer número cuenta de a 9, se mueve cada 9 aplausos por lo tanto hay que multiplicarlo por 9.
Entonces [a,b,c]---> a*9+b*3+c (o [a,b,c]--->a+b*3+c*9)

* Resolución esquemática del problema de pasar de un número de aplausos a la posición de los estudiantes:

Opción 1

Si el número es mayor o igual a 9 (pero menor a 26, si no no se puede representar con 3 cifras) se divide entre 9, el cociente es la última cifra. El resto, que es menor a 9 se lo divide entre 3 y el cociente es la penúltima cifra, el resto que es menor a 3 son las unidades.

Si es menor a 9 la ultima cifra es 0 y se divide al número entre 3 y el cociente es la segunda cifra y el resto las unidades. A continuación usamos la siguiente notación, para cociente (//) y resto (%) de la división entera respectivamente, que es la misma que usa python para esas operaciones.

Si n>=9 la ultima cifra es n//9 (cociente de la división entera de n entre 9), la segunda cifra es (n%9)//3 (el cociente de la división entera del resto de n entre 9 entre 3), la primera cifra es (n%9)%3 (el resto de la división anterior).

Si n<9 la última cifra es 0, la segunda es n//3 y la primera es n%3.

Opción 2

Se divide el número entre 3 y el resto son las unidades, al cociente se lo divide entre 3 y el resto es la segunda cifra. El cociente, que es menor a 3 es la tercer cifra.

Primera cifra n%3
Segunda cifra (n//3)%3
Tercera cifra (n//3)//3

La última opción es ciertamente más elegante, sobre todo si se observa que la tercer cifra también es ((n//3)//3)%3, el problema es que es menos intuitiva. En todo caso es muy difícil predecir el algoritmo exacto que los estudiantes van a desarrollar, este es sólo un ejemplo, la siguiente demostración es importante para el curso de 2°BD si se toma como objetivo formalizar razonamientos.

Proposición: Ambos métodos son equivalentes

n=3q1+r1  r1<3
q1=3q2+r2  r2<3
q2=3 0+r3  r3=q2<3

De la opción 2 a la opción 1
==>
n=3*(3q2+r2)+r1=3*(3*(r3)+r2)+r1=9r3+3r2+r1 ALQQL (a lo que queríamos llegar)

De la opción 1 a la opción 2

Además si n= 9r3+3r2+r1 ==> n//9 =r3 ya que 3r2+r1<9 y (3r2+r1)//3= r2 y (3r2+r1)%3=r1
Por lo tanto ambos métodos son equivalentes.
Algoritmos en python2

```
 # Pasamos el número de aplausos a la posición de los estudiantes, con el algoritmo de la opción 1
 def posiciones1(número):
        lista=[0,0,0]
        if número>=9:
                lista[2]= número // 9  # recordar, número // 9 es el cociente de dividir número entre 9
                lista[1]= (número % 9) // 3  #recordar, número % 9 es el resto de dividir número                                                    #entre 9
                lista[0]= (número % 9) % 3
        else:
                lista[1]=número // 3
                lista[0]=número % 3


 # Pasamos el número de aplausos a la posición de los estudiantes, con el algoritmo de la opción 2

 def posiciones2(número):  #ciertamente esta opción es mas eficaz, simple y general.
        lista=[]
        while número > 0:
                lista=lista+[número % 3]
                número=número // 3
        return lista

 # Escribimos la posición y nos devuelve el número de aplausos

 def aplausos(l):
        n=0
        estudiante=0
        while estudiante<3:
                n=n+l[estudiante]*(3**estudiante)
                estudiante = estudiante + 1
        return n
```

* Otros algoritmos para trabajar con cualquier base.

Incluimos la implementación en python de algoritmos para pasar un número en base 10 a otra base cualquiera y viceversa.

```
 # Tomamos una base y un número en base 10, y devuelve una lista con las cifras del número en la base dada.

 def diez_a_base(número_10,base):
        lista=[]
        while número_10 > 0:
                lista=lista+[número_10 % base]
```

```
                número_10=número_10 // base
        return lista
 # Tomamos una base y una lista que contiene las cifras en esa base, y devuelve un número en base 10

 def base_a_10(l,base):
        n_10=0
        recorre=0
        while recorre<len(l):
                n_10=n_10+l[recorre]*(base**recorre)
                recorre = recorre + 1
        return n_10
```

* Adición de números en base 3

Para terminar con esta secuencia, se incluye un diseño de algoritmo de suma de números en base 3 y una implementación en python.

ESPECIFICACIÓN DEL PROBLEMA:

Entrada: Una dupla (sumando, sumando) escritos como listas de cifras.
Salida: Suma de sumando1 y sumando2 escrito como una lista de cifras.

Primero,
Se generan 2 listas de igual tamaño que los sumandos de tal forma que las cifras faltantes se rellenan con ceros.
Luego, se calcula cada una de las cifras del resultado de la siguiente manera:

La primera cifra del resultado, la de las unidades, es el resto de dividir entre 3 (la base) a la suma de las primeras cifras de cada uno de los sumandos. El cociente de esta división es el acarreo (las decenas que me llevo), que se guarda.

La segunda cifra es el resto de dividir entre 3, a la suma de las cifras de la segundas cifras de cada sumando y el acarreo. Y se redefine el acarreo como el cociente de esta división
Las cifras desde la tercera en adelante se calculan igual que la segunda.

De hecho si se define el acarreo como 0 al comenzar la suma, entonces, la i-esima cifra de la suma se calcula sumando la i-esima cifra de cada sumando más el acarreo.

Por último si el acarreo, al final, no es 0 se agrega una cifra al resultado colocando en ella el último acarreo.

Nota: Para python el primer elemento de una lista es el que está a la izquierda, sin embargo en la numeración posicional los números se escriben de izquierda a derecha. Por eso en la implementación python el primer paso es invertir las listas y al final antes de dar la respuesta se invierte nuevamente.

## Implementación python

```
# Tomamos una dupla (sumando, sumando), y devuelve la suma de la adición de sumando1 y sumando2 escrito como una lista
# Como
def suma(l1,l2):
        l1=l1[::-1]                     #Invertimos las listas
        l2=l2[::-1]                     # de los sumandos
        lnorm=[]                        #lnorm es la lista rellenada con ceros (normalizada)
        suma=[]
#Escribimos ambas listas con igual número de elementos
# rellenamos los lugares faltantes con 0 a la izquierda
        i=1
        while i <= max(len(l1),len(l2)):
                lnorm=lnorm+[0]
                suma=suma+[0]
                i=i+1
        if len(l1)<len(l2):
                for cifra in range(0,len(l1)):
                        lnorm[cifra]=l1[cifra]
                l1=lnorm
        else:
                for cifra in l2:
                        lnorm[cifra]=l2[cifra]
                l2=lnorm

# Adición con algoritmo escolar.
        acarreo=0
        for cifra in range(0,len(l1)):
                suma[cifra]=(l1[cifra]+l2[cifra]+acarreo)%3
                acarreo=(l1[cifra]+l2[cifra]+acarreo)//3
        if acarreo!=0:
                suma=suma+[acarreo]
        return suma[::-1]  #invertimos el resultado final al devolverlo.
```

## Conclusiones:

En esta secuencia se evitó explícitamente, una introducción básica en python y se enfocó el trabajo en la parte matemática, por lo tanto no es recomendable usarla como primera actividad para el trabajo en python, de hecho se puede eliminar completamente la implementación en este lenguaje y la actividad no cambiaría en su esencia. Sin embargo trabajar con python resulta útil fundamentalmente en dos aspectos:

Como motivación, no sólo por trabajar con las máquinas si no como actividad creadora, ya que generar un programa que funciona es muy satisfactorio. La implementación en python puede ser el cierre para la resolución de un problema algorítmico.

Como un actor que obliga al estudiante a explicitar el algorítmo con rigurosidad, y validador del resultado. Ya que el estudiante puede probar su programa y verificar por si mismo si funciona bien o funciona mal generando autonomía en el estudiante [1]. Esto fomenta que el docente adopte un rol de guía para el estudiante, y que éste sea el principal agente de la construcción de su conocimiento.

## Bibliografía

1. . Gilles Dowek, Quelle informatique enseigner au lycee?, https://who.rocq.inria.fr/Gilles.Dowek/lycee.html (2005)

# CÉDULA DE IDENTIDAD. DÍGITO DETRÁS DEL GUIÓN

Luis Langon (Liceo Solymar 2 , Canelones)

Todos conocemos nuestro número de cédula y sabemos algo del número de otra persona... Es fácil suponer que se trata de un número de 7 cifras (tal vez 6 en una persona de mayor edad) seguido por un guión y otro número entre 0 y 9.

No es frecuente que nos preguntemos ¿para qué sirve este último dígito?... si es que sirve para algo... Se lo conoce con el nombre de "el dígito verificador" está claro entonces que sirve para verificar. Para verificar que los números anteriores han sido bien escritos.

¿Cómo con un número solo se puede saber que los 7 anteriores fueron bien o mal escritos? Bueno ahí es donde entra la matemática... Ya está todo pronto para empezar a trabajar e investigar sobre este tema.

* ¿Cómo se calcula ese dígito?

Veremos con un ejemplo cómo es este método [1].

Se necesita una cédula, por ejemplo 3.416.652-_ y el número 2987634, que es una constante dada (a la cual se le quita el primer 2 en caso de que la cédula tenga 6 dígitos).

Primer paso: Se multiplica cada dígito de la cédula por cada dígito del número dado y se anota la última cifra de cada resultado.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 3 | 4 | 1 | 6 | 6 | 5 | 2 |
| x | 2 | 9 | 8 | 7 | 6 | 3 | 4 |
| | 6 | 6 | 8 | 2 | 6 | 5 | 8 |

Segundo paso: Se suman todos los números que se anotaron en el paso anterior y se anota la última cifra de esa suma.

6 + 6 + 8 + 2 + 6 + 5 + 8 = 41 → 1

Tercer paso:

Si es cero, entonces el dígito verificador es 0.

Si no es cero debe calcularse la diferencia con 10.

10 – 1 = 9 el dígito verificador es 9.

1 "Una introducción a los códigos detectores de errores y correctores de errores" Pablo Fernández Gallardo y Omar Gil

## Objetivo

Trabajar con las operaciones básicas, suma, multiplicación y división entera (en este caso, entre 10), siendo utilizado tanto el resto como el cociente de la división entera, se descubrirán algunas propiedades de la división entera entre 10.

Se pretende dar acá los "primeros pasos en programación" (entendiendo programación como la habilidad de escribir un programa que implemente una solución de un problema). Son los siguientes:

Saber especificar el problema que se tiene como un problema algorítmico (puede darse ejemplos de problemas no algorítmicos).

Encontrar una solución y saber diseñarla en forma de algoritmo, o sea una transformación de los datos de entrada en la salida deseada, escrita en forma de pasos.

Saber traducir dicha solución a un programa, para eso se deberán conocer los comandos básicos de entrada, salida, asignación, bucle y condicional y habrá que trabajar la sintaxis específica del lenguaje python que es el que se usa en esta pasantía (ver manual adjunto).

Nivel: El bloque didáctico está pensado para primeros años de ciclo básico, pero es posible llevarlo adelante en otros niveles ya que el objetivo es utilizarlo como ejemplo de algoritmo y dar los primeros pasos en programación e implementación.

## Actividad1

Presentar el tema y el objetivo

¿Por qué tenemos tal número de cédula? ¿Es por edad?

¿Por qué hay un número separado por un guión? ¿Sirve para algo?

Hacemos 10 grupos, uno para cada dígito, juntando en cada grupo a los alumnos que tienen el mismo dígito en su cédula. Damos tiempo para que prueben algoritmos con los números de sus cédulas y que el resultado sea su dígito verificador... Dejamos que prueben diferentes estrategias pero al final... ¡No encontramos nada! Los alumnos buscarán la información acerca del significado de ese número y de cuál es el algoritmo correcto para calcularlo (esta información está disponible en internet).

## Actividad 2

Los alumnos deberán escribir las instrucciones que se siguen para calcular el dígito de control. Cada grupo debe redactar un procedimiento para que los otros sigan las instrucciones que llevan a calcular el dígito verificador. Se practica esto en forma de juego donde cada equipo sigue las operaciones que el otro equipo dio. Formalizamos el concepto de algoritmo en base a lo estamos haciendo.

"Entrada (ci) ---> algoritmo ---> salida(d)"

## Actividad 3

Aquí se lleva a cabo el trabajo sobre la computadora, se busca que los alumnos implementen y especifiquen este algoritmo en lenguaje python utilizando el editor gedit y ejecuten el programa en un terminal.

Para que puedan hacerlo se deben aclarar ciertos puntos:

La carpeta en la que graban el archivo ejemplo: *"/estudiante/Documentos"*

El nombre del archivo al guardar por ejemplo: *"cédula.py"*
Ejecutar el terminal en el menú: *"Aplicaciones > Accesorios > Terminal"*
Seleccionar la carpeta correcta escribiendo: *"cd /estudiante/Documentos"*
Ejecutar el intérprete python escribiendo: *"python"*
Ejecutar el programa escribiendo: *"import python"*
Listar y aclarar los comandos python que se necesitan, que se encuentran explicados en el manual adjunto:

Asignación ejemplo: *"m = 2987634*"
Repetición: *"while"*
Condicional: *"if"*
entrada: *"input"*
salida: *"print"*

* Especificación

Entrada: Número entero (cédula sin puntos, ni guión, ni dígito verificador) llamarlo *ced.*
algoritmo: Considerar el número entero 2987634, llamarlo *m*.
Sea *suma* = 0 una variable que utilizaremos para ir sumando.
Repetir estos pasos mientras queden cifras disponibles en *ced*
multiplicar el último dígito de la cédula por el último de *m.*
(esto sería *ced* mod[1] 10 * *m* mod 10)
agregarlo a la suma.
quitar el último dígito de *ced* y de *m.*
(esto sería *ced* = *ced* div[2] 10 y m = m div 10)
Al terminar con todas las cifras de la cédula.
Tomar en *dig* la última cifra de *suma (suma mod 10)*.
Si es "0" entonces
devuelva 0 en *dig*
si no...
devuelva (10 - *suma*) en *dig*
Salida: *dig* número entero (dígito verificador de la cédula ingresada)

* Implementación en Python

```
# funcion que calcula el digito despues del guion
# entrada: Cédula número entero, (hasta 7 cifras)
# salida:  DIGITO número entero de un digito
def digito(ci):
    m = 2987634        # número magico
    d = 0              # digito de la cédula
    while m > 0:
```

1 Mod: módulo, es el resto de la división entera.

2 Mod: módulo, es el resto de la división entera.

```
                n = ci % 10     # último digito de la cédula
                ci = ci // 10   # quito el último digito
                d = d + m%10 * n
                m = m // 10
        d = d % 10
        d = 10 - d
        if d == 10:
                return 0
        else:
                return d

# para probar la funcion
print "Ingrese su número de cédula."
print "Sin puntos, ni guion, ni digito de control"
print "Escriba 0 para terminar"
ci = input ("cédula: ")
while ci != 0:
        print digito (ci)
        ci = input ("cédula: ")
```

## Conclusión

Ha sido un gran placer estudiar programación y conjugarlo con la enseñanza de la matemática. Pensando en el potencial que estas estrategias puedan llegar a tener para enseñar matemáticas y para entusiasmar al alumnado -que vemos cada vez más alejado del estudio tradicional-, sentimos que se ha encontrado una nueva herramienta en la computadora -que no estaba claro cómo aprovecharla- en *"la programación en la enseñanza de la matemática"*. La secuencia didáctica que se propuso pretende ser un estímulo positivo para que el estudiante trabaje, para que se enfrente a un reto con entusiasmo y vea que sus objetivos se van cumpliendo a medida que logra desarrollar algoritmos correctos que le permiten resolver los problemas y sienta una enorme satisfacción por haberlo conseguido. Nosotros mismos nos tropezamos con múltiples inconvenientes en el proyecto y cada obstáculo que se sorteaba nos daba tal satisfacción que se generaban las fuerzas para seguir hacia adelante.

# PROPUESTA SOBRE CONJUNTOS

Santiago Martorell y Patricia Añón - CETP, CES

## FUNDAMENTACIÓN

El ejercicio que se propone está pensado para trabajar con relaciones y operaciones básicas de conjuntos (unión, intersección, inclusión y diferencia). El abordaje intenta fusionar conceptos básicos de teoría de conjuntos que pueden ser trabajados en diversas asignaturas (matemática, lógica) del liceo y/o utu, y conceptos de programación básicos y necesarios en la enseñanza de la informática. De esta manera, la teoría (conceptos puntuales a trabajar) y la práctica (escritura del programa) se unifican en el ejercicio, intentando apartarnos de la idea de dicotomía entre uno y otro.

Se trata de diseñar un algoritmo para cada una de las operaciones e implementarlo como un programa en el lenguaje python. Si bien podemos encontrar funciones preestablecidas en python para cada una de las operaciones binarias con conjuntos, la idea central es que los estudiantes elaboren e implementen sus propios algoritmos como forma de reforzar la comprensión de los conceptos involucrados, útiles tanto para matemática como para programación.

## PROPUESTA DE TRABAJO

Se propondrá al estudiante el siguiente ejercicio:

> Dados dos conjuntos de números enteros, diseña algoritmos que devuelvan el resultado de cada una de las siguientes relaciones y operaciones:
> - Unión
> - Intersección
> - Inclusión
> - Diferencia
>
> Implementa en python cada uno de los algoritmos.

El programa principal, dónde se ingresan los elementos de los conjuntos y se elige la operación a realizar, es dado por el docente (lo encontrará en el apéndice). Los estudiantes deben elaborar un módulo con cada una las operaciones implementadas y usar el programa dado.

Para llegar a la solución, necesariamente habrá un proceso de pensamiento sobre el problema que se plantea, el algoritmo que nos lleva a la misma y los pasos a seguir para determinar cada una de las operaciones que el programa debe realizar.

Otro aspecto de fundamental importancia, es la interacción que se da entre el lenguaje formal y el lenguaje natural, dado que se le pide al alumno que lo exprese en lenguaje natural y/o en pseudocódigo. De esta manera se le está dando la posibilidad de que se concentre en el contenido y no tanto en utilizar un determinado lenguaje de programación para expresarlo. Una vez que ha pensado suficientemente los pasos para resolver el problema en el lenguaje natural, podrá concentrarse en escribirlo en un lenguaje de programación, instancia que presenta la dificultad del dominio de dicho lenguaje, pero que no tiene que ver con las dificultades en la solución misma del problema.

## DESCRIPCIÓN DEL ALGORITMO

Antes de intentar escribir en un lenguaje de programación la solución al problema planteado, es conveniente especificar el problema, diseñar la solución en lenguaje natural, pensar qué pasos debemos seguir para llegar a la misma, dejando para más adelante la dificultad de la implementación. Esto es, resolver el problema en sí, sin que en ello influyan dificultades que puedan tener que ver con el dominio del lenguaje de programación.

El pseudocódigo utilizado es una mezcla de lenguaje natural (español) con lenguaje matemático. Constituye una guía para confeccionar el código final, por ello es conveniente tener presente que no es necesario profundizar en la definición de variables, estructuras, y otros elementos, durante la construcción del mismo. Probablemente, al momento de confeccionar el código final, se modificarán algunas partes del pseudocódigo, ya que hay que adaptarse a las limitaciones que tienen los lenguajes de programación, en cuanto a estructuras disponibles y sintaxis, entre otros.

* Operaciones a realizar:

- Dados dos conjuntos A y B, determinar el conjunto C resultado de realizar la unión entre ambos:
  - Si A es vacío
    - C = B
  - Si no, si B es vacío
    - C = A
  - o Si no
    - C = A
    - Mientras tenga elementos en B, tomo elemento de B
      - Si elemento no pertenece a C
        - C = C + elemento de B
  - Devuelvo C
- Dados dos conjuntos A y B, determinar el conjunto C resultado de realizar la intersección entre ambos:

- Si A o B vacíos
  - C = Ø
- Si no
  - Mientras tenga elementos en A, tomo elemento de A
    - Si elemento de A pertenece a B
      - C = C + elemento de B
- Devuelvo C

• Dados dos conjuntos A y B, determinar el conjunto C resultado de realizar la diferencia entre ambos (A – B):

- Si A vacío
  - C = Ø
- Si no
  - C = A
  - Mientras tenga elementos en B, tomo elemento de B
    - Si elemento de B pertenece a C
      - C = C – elemento de B
- Devuelvo C

• Dados dos conjuntos A y B, determinar si A está incluido en B, devolviendo un booleano como resultado:

- Resultado = TRUE
- Si cantidad de elementos de B es mayor o igual a cantidad de elementos de A
  - Mientras tenga elementos en A, tomar elemento de A
    - Si elemento no pertenece a B
      - Resultado = FALSE
- Si no
  - Resultado = FALSE
- Devolver Resultado

## Módulo con implementaciones en Python de relaciones operaciones y con conjuntos

```
import copy

def intersec(c1, c2):
        resultado = []
        for valor in c1:
                if pertenece(c2, valor):
                        resultado.append(valor)
        return resultado

def inclusion(c1, c2):
        resultado = True
        contador = 0
        while resultado and contador < len(c1):
                resultado = pertenece(c2, c1[contador]) and resultado
                contador +=1
        return resultado

def union(c1, c2):
        if len(c1)== 0:
                resultado = c2
        elif len(c2)== 0:
                resultado = c1
        else:
                resultado = copiar(c1)
                for valor in c2:
                        pertenececc1 = pertenece(c1, valor)
                        if not pertenececc1 :
                                resultado.append(valor)
        return resultado

def copiar(c1):
        resultado = []
        for valor in c1:
                resultado.append(valor)
        return resultado

def mostrarconjuntos(listaconjuntos):
        contador = 1
        for conjunto in listaconjuntos:
                print "c",contador,": ",conjunto
                contador +=1

def pertenece(c1, número):
        resultado = False
        contador = 0
```

```
        while contador < len(c1) and not resultado:
                if c1[contador] == número:
                        resultado = True
                contador +=1
        return resultado

def diferencia(c1, c2):
        resultado = []
        for valor in c1:
                if not pertenece(c2, valor):
                        resultado.append(valor)
        return resultado
```

* Aclaraciones sobre la implementación

Es apropiado, para la implementación en python, introducir algunos elementos de este lenguaje. Se resumen a continuación:

- **For:** repite un segmento de código, dado un dominio de n elementos, el segmento de código que se encuentra en el bloque se repetirá n veces.
- **Append:** agrega un elemento a un conjunto de elementos.
- **And:** operador "y" lógico.
- +=1: le suma uno al valor actual de la variable y se lo asigna a la misma.
- ==: operador de igualdad, compara dos valores, devuelve TRUE si son iguales, FALSE si no.
- **If:** evalúa una expresión booleana, y ejecuta sentencias de acuerdo a su valor.
- **Return:** devuelve el resultado de una función.
- **Not:** dada una expresión, realiza la operación de negación de la misma. Si la expresión se evalúa TRUE, NOT expresión se evaluará FALSE.

Para ampliar esta información, ver el manual que complementa este trabajo.

En el apéndice se proporciona un programa principal para que el profesor entregue a los estudiantes, de forma tal que estos puedan probar sus funciones implementadas en python. Para ello se debe proceder de la siguiente manera:

- Crear módulo para programa principal (con extensión .py).
- Ingresar el código brindado.
- Tener en cuenta el nombre de las funciones en el programa dado (diferencia, intersec, union, inclusion); si se utilizan otros nombres deben ser modificados en el programa principal.

## EJERCICIOS

Se presentan a continuación algunos ejercicios, con los que se puede comenzar a probar el programa. A través de casos puntuales se guía al estudiante para llegar a enunciar propiedades y/o generalizaciones, comenzando por expresar las mismas en el lenguaje natural,

para luego formalizarlas en lenguaje simbólico.

Además de continuar trabajando la temática, podrán utilizar el programa de cuya creación fueron partícipes para seguir reafirmando conceptos de teoría de conjuntos.

1) Dados los siguientes conjuntos:

- $A = \{2,4,6,8,9\}$ y $B = \{3,4,5,9\}$
- $M = \{10,12,15,19\}$ y $N = \{21,25,1\}$
- $W = \{6,9,5,34,10,20\}$ y $R = \{34,5,9\}$

Resuelva:

- $C = A (A \cap B)$
- $P (M \cap N)$
- $Z(W \cap R)$

¿Se cumple lo siguiente? Justifique (puede utilizar el programa para sacar sus conclusiones):

- $C \subseteq A$ y $C \subseteq B$.
- $P \subseteq M$ y $P \subseteq N$.
- $Z \subseteq W$ y $Z \subseteq R$.

2) Dadas las siguientes igualdades:

- $(C1 \cap C2) \cup C1 = C1 \cup C2$
- $(C1 \cap C2) \cup C1 = C1$
- $(C1 \cap C2) \cup C1 = C2$

a. Marque la opción correcta y justifique en todos los casos. (Puede corroborar que la opción tomada es la correcta, utilizando el programa).

b .De acuerdo a lo resuelto en el punto a), ¿podría llegar a una generalización? En caso afirmativo, exprésela en lenguaje natural.

c. Complete los espacios punteados:

$\forall A , \forall B$, si $\ldots \subseteq A$ entonces $A \cup B \ldots A$

## CONCLUSIONES

Se espera que el estudiante pueda, en principio, lograr diseñar los algoritmos necesarios para cada relación y operación. Esta actividad, no solo permite ver si el concepto abordado (la operación en cuestión) fue comprendida, si no también enfocarse en donde se presentan las dificultades para poder superarlas.

Al mismo tiempo, al implementarlo en python, el educando puede aproximarse a la programación partiendo de un conocimiento matemático, y vivenciar el "rigor" al enfrentar los

errores que se marcan en la compilación del programa, sin que sea el docente el encargado de marcarlo si no la objetividad de una máquina.

Además, se espera que el programa, en parte creado por cada uno de los estudiantes, pueda ser puesto en uso para resolver problemas como los propuestos anteriormente, y continuar reflexionando sobre la teoría de conjuntos, con un producto creado por ellos mismos.

Se espera que mediante esta actividad el estudiante contribuya a desarrollar una nueva forma de trabajo, basándose en el pensamiento algorítmico, y que sea capaz de aplicarlo en cualquier ámbito en el que resulte eficaz. Esto refiere a generar, a propiciar, mediante el lugar que nos toque en el ámbito educativo, un modelo de pensamiento en el que se hace énfasis en la importancia de la reflexión del paso a paso a seguir para llegar a resolver un problema, disminuyendo así las posibilidades que existen de cometer errores.

El hecho de que se utilice un lenguaje de programación, en la etapa final, permite que se alcance un grado de formalización importante, surgido de la necesidad de seguir la sintaxis y la semántica del mismo.

Por otra parte, la posibilidad de ejecutar el programa, y observar su comportamiento, genera una motivación extra, por lo que se vuelve un objeto didáctico ventajoso.

Finalmente, cabe destacar que el desarrollo de este tipo de actividades, contempla la interdisciplinariedad, ya que se trabajan aspectos matemáticos, lógicos, y del lenguaje. Esto es importante, ya que es una manera de enlazar distintas disciplinas desde un trabajo puntual, y evidenciar cuán enriquecedor resulta comprender que el conocimiento no se limita a un área en particular, si no que es, en el entramado de áreas que se vinculan entre sí.

## *APÉNDICE*

### Implementación en python del programa principal

Se brinda a continuación, un posible programa principal, comentado (el texto que figura luego del # es el comentario de la línea en que se encuentra), de forma que puedan ser probadas las funciones diseñadas y programadas por los estudiantes.

```
import opconj
contadorconj = 0 # variable para contar la cantidad de conjuntos que se han ingresado
contador = 0 # variable para contar de uso general
respuesta = 1 # variable para seleccion de menu principal
listaconjuntos = [] # arreglo para almacenar los conjuntos (en cada posición se almacena un conjunto)

conjuntoA = 1 # variable que indica el número del primer conjunto con el que se trabaja
conjuntoB = 2 # variable que indica el número del segundo conjunto con el que se trabaja

#cargo conjuntos de prueba
listaconjuntos.append([1,2,3,4,5,6,7]) #primer conjunto
listaconjuntos.append([1,2,3,4,5,6,7,8,9]) #segundo conjunto
```

```
while respuesta > 0: #Comienzo del menu principal - con opción 0 termina el programa
        print
 "#############################################################"
        print "Conjuntos"
        print "1-Ingresar conjuntos"
        print "2-Operaciones binarias"
        print "0-Salir"
        respuesta= int(raw_input("Ingrese una opción: "))
        print
 "#############################################################"
        if respuesta == 1: #seleccion de opción 1 del menu principal - Ingresa conjuntos
                listaaux = []
                print "Ingresar conjuntos"
                salir = 'n' #inicializo la variable salir
                while salir != 's': #pide elementos y los agrega al conjunto mientras no se ingrese la
letra "s"
                        salir = raw_input("Ingrese un elemento (s para terminar): ")
                        if (salir != 's' and salir != 'n'): #chequea que lo ingresado no sea ni "s" ni "n"
                                número=int(salir) #se transforma el texto a número
                                if opconj.pertenece(listaaux, número) == False: #se chequea que no
pertenezca ya al conjunto
                                        listaaux.append(número) #se agrega el elemento al conjunto
                                else:
                                        print "El elemento ya pertenece al conjunto"
                listaaux.sort() #se ordena el conjunto
                listaconjuntos.append(listaaux) #se agrega el nuevo conjunto a la lista de conjuntos
                print "El conjunto ingresado es: ", listaaux #se muestra el conjunto ingresado
                contadorconj +=1 #se aumenta la cantidad de conjuntos disponibles en 1
                opconj.mostrarconjuntos(listaconjuntos) #se muestran todos los conjuntos
        elif respuesta == 2: #seleccion de opción 2 del menu principal - Operaciones binarias
                respuesta2 = 1
                while respuesta2 != 0: # Comienzo del menu de operaciones binarias - con opción 0
vuelve a menu principal
                        print
 "#########################################################"

                        print "Operaciones binarias"
                        print "1-Interseccion"
                        print "2-Inclusion"
                        print "3-Union"
                        print "4-Diferencia"
                        print "5-Ver Conjuntos"
                        print "6-Seleccionar Conjuntos"
                        print "0-Salir"
                        respuesta2 = int(raw_input("Ingrese una opción: "))
                        print
 "#########################################################"
```

```
                if respuesta2 == 1: #selecciona opción 1, se realiza la interseccion entre los conjuntos seleccionados
                        #llamo a la funcion que realiza la interseccion y muestro el conjunto resultado
                        print "La interseccion es: ",opconj.intersec(listaconjuntos[conjuntoA-1], listaconjuntos[conjuntoB-1])
                elif respuesta2 == 2: #selecciona opción 2, chequea la inclusion del conjuntoA en el conjuntoB
                        #llamo a la funcion que chequea la inclusion y muestro el resultado
                        resultado = opconj.inclusion(listaconjuntos[conjuntoA-1], listaconjuntos[conjuntoB-1])
                        if len(listaconjuntos[conjuntoA-1])==len(listaconjuntos[conjuntoB-1]) and resultado:
                                print "c",conjuntoA,": ",listaconjuntos[conjuntoA-1], " esta incluido estrictamente en c",conjuntoB,":", listaconjuntos[conjuntoB-1]
                        elif resultado:
                                print "c",conjuntoA,": ",listaconjuntos[conjuntoA-1], " esta incluido en c",conjuntoB,":", listaconjuntos[conjuntoB-1]
                        else:
                                print "c",conjuntoA,": ",listaconjuntos[conjuntoA-1], " NO esta incluido en c",conjuntoB,":", listaconjuntos[conjuntoB-1]
        elif respuesta2 == 3: #selecciona opción 3, se realiza la union entre los conjuntos seleccionados
        #llamo a la funcion que realiza la union y muestro el conjunto resultado
                        print "La union es: ", opconj.union(listaconjuntos[conjuntoA-1], listaconjuntos[conjuntoB-1])
        elif respuesta2 == 4: #selecciona opción 4, se realiza la diferencia entre los conjuntos seleccionados
        #llamo a la funcion que realiza la diferencia y muestro el conjunto resultado
        print "La diferencia es: ", opconj.diferencia(listaconjuntos[conjuntoA-1], listaconjuntos[conjuntoB-1])
        elif respuesta2 == 5: #selecciona opción 5, se muestran los conjuntos disponibles
                #llamo a la funcion que muestra la lista de conjuntos disponibles
        opconj.mostrarconjuntos(listaconjuntos)
        elif respuesta2 == 6: #selecciona opción 6, se seleccionan los conjuntos para trabajar
        print "Los conjuntos seleccionados actualmente son: c",conjuntoA," y c",conjuntoB
        opconj.mostrarconjuntos(listaconjuntos) #muestro conjuntos disponibles
        #se piden los nuevos conjuntos (se deben ingresar solo números)
        conjuntoA = int(raw_input("Ingrese el número correspondiente al primer conjunto: "))
        conjuntoB = int(raw_input("Ingrese el número correspondiente al segundo conjunto: "))

        #cierre del bloque condicional del menu de operaciones binarias
                #cierre del while del menu de operaciones binarias
        #cierre de bloque condicional del menu principal
#cierre del while principal (termina el programa)
```

# DIVISIBILIDAD

Teresa Pérez (Liceo Solymar 1, Canelones, CES)

El objetivo de la secuencia es ***en forma simultánea,*** estudiar algunos conceptos básicos relativos a divisibilidad e introducir nociones de programación utilizando como lenguaje phyton.

Las nociones básicas de divisibilidad no son nuevas para los estudiantes liceales - ya sea de 1°CB, como de 2°BD - lo que lo hace especialmente adecuado para está introducción *"en simultáneo"* de las nociones de algoritmia, que sería el verdadero contenido nuevo. Creemos a su vez que ayudará a lograr una conceptualización más profunda de contenidos matemáticos con los que los estudiantes ya están familiarizados desde un nivel fundamentalmente procedimental.

En principio las actividades están pensadas tanto para 1°CB como para 2°BD (específica), lo que sin duda será diferente son las producciones de los estudiantes y el nivel de profundización y fundamentación que se llegue a lograr.

Se detallarán tres actividades, y luego se agregan otras como continuación del trabajo que pretende integrar nociones de algoritmia, a través del estudio del tema divisibilidad en la clase de matemática.

## Actividad 1:

*Se hace correr un programa (acertijo1) [1] que pide a los estudiantes un número natural y les devuelve el conjunto de divisores del mismo.*

*La actividad consiste en [2]:*

- *Experimenta varias veces con el programa.*
- *Explica para qué sirve el programa y cámbiale el nombre de acuerdo a ello.*
- *Escribe qué datos se debe proporcionar y qué información devuelve el programa.*
- *Escribe los pasos de un procedimiento que te permita hacer lo mismo que el programa.*

1 En caso de problemas al abrir el archivo, ver el manual.

2 Cada docente decidirá cómo dosifica la consigna de acuerdo a las características de sus grupos, si van todas juntas de a una, según el ritmo de cada uno

El propósito de está actividad es desde el punto de vista de los conceptos matemáticos, introducir la noción de conjunto de divisores, que el estudiante conoce desde primaria. Desde el punto de vista de la programación, se busca en primer lugar generar curiosidad acerca de cómo la máquina resuelve el problema de hallar el conjunto de divisores de un número natural.

Luego de que el grupo reconozca de qué problema se trata, se formalizará la noción de conjunto de divisores, para la cual se hace necesaria alguna definición de divisor. La definición elegida, dependerá de los procedimientos que hayan surgido en el grupo para resolver el problema de encontrar el conjunto de divisores de un número. Inclusive es posible que surja más de una opción, y puede ser interesante aprovechar la instancia para trabajar las relaciones entre diferentes definiciones de un objeto matemático.

El tener que descubrir lo que hace el programa y luego elaborar una estrategia personal, le hace no solo poner en juego conceptos matemáticos, si no ordenar en forma de secuencia temporal los procedimientos que utiliza para resolver el problema, o sea desarrollar un algoritmo.

Si nos centramos en la introducción a la algoritmia[1], se busca en primer lugar que el alumno pueda formular el problema que resuelve la máquina, especificando las condiciones de entrada y de salida. Estos conceptos deben explicitarse en la puesta en común en la clase. Para el trabajo en la construcción de procesos algorítmicos es clave, ya que la especificación de las condiciones de entrada y salida son el primer paso para comprender un problema algorítmico [2]. Desde el punto de vista de los contenidos matemáticos, el determinar con claridad que para *"Hallar los divisores de un número natural"* para cada número natural (el objeto de entrada es un número) obtengo un conjunto de números naturales (el objeto de salida es un conjunto) mejora sin duda la conceptualización de la noción "conjunto de divisores", pero además y en forma no explícita, ayuda a la construcción de la noción de función, que si bien no es un objetivo visible, resulta fundamental en los procedimientos de programación.

Lo que realiza el programa, es en principio una "caja negra", por eso se pide al estudiante que explicite posibles algoritmos que devuelvan el mismo resultado que el programa. La discusión de los distintos procedimientos resulta fundamental para la construcción de un pensamiento algorítmico. Se sugiere como actividad de clase intercambiar los procedimientos, y que los estudiantes deban "ejecutar" algún procedimiento pensado por otro compañero.

El último paso, sería mostrar una posible implementacion en python, de modo de ir introduciendo el mundo de los lenguajes de programación y algunos elementos de la implementación.

En definitiva: el programa no es otra cosa que una secuencia ordenada de pasos, escrita en un determinado código, que debe ser preciso y correcto para poder ser entendido por la máquina. El programa lo escribo como un texto y luego necesito un "intérprete" que lo ejecute.

---

1 Es en este punto donde se conjugan el aspecto matemático del problema y su vinculación con la programación, que va más allá de las cuestiones de implementación.

Además permite que el alumno se familiarice con algunos aspectos de la sintaxis de python.

Desarrollaremos la solución al problema de obtener el conjunto de los divisores de un número natural dado y luego su implementación. Eventualmente se puede pedir que los estudiantes lo ejecuten manualmente.

Para ello primero debemos analizar si el mismo está realmente formulado como un problema algorítmico. ¿Están especificadas las condiciones de entrada? ¿Están especificados los elementos de salida deseados?

**# Especificación de la entrada**: un número natural
# **Especificación de elementos de salida:** el conjunto de los divisores del número de la entrada

Pasaremos a desarrollar una solución del problema, o sea un algoritmo, en español y en python. (implementación en phyton) [1]

| ***Alogaritmo en español*** | ***Implementación en phyton*** |
|---|---|
| Escribir un número natural *a*.<br>Para cada natural entre 1 y *a*, que llamaremos *x,* calcular el resto de dividir *a* entre x.<br>Si el resto obtenido es 0, escriba *x* en la lista *d*<br>Escriba la lista *d*. | a = input("escriba un número natural: ")<br>for x in range(1, a+1) a%x<br>d = [x for x in range(1, a+1) if a%x= =0]<br>print "d =", d |

Si observamos el segundo y tercer pasos del procedimiento:

- Para cada natural entre 1 y a, que llamaremos x, calcular el resto de dividir a entre x. Se escribe en python: for x in range(1, a+1) a%x
- Si el resto obtenido es 0, se escribe en python: if a%x == 0[2]
- Finalmente la lista d por comprensión es: d = [x for x in range(1, a+1) if a%x==0]

Se lee: d es la lista de los elementos x, $1 \leq x \leq a$, tales que a%x == 0

En este contexto, la definición matemática más adecuada para divisor de un número es: a|b <=>bmod(a)=0 [3] y para conjunto de divisores de a: {x tales que x|a }

1 En la versión que ejecutan los estudiantes se agregaron espacios para hacer más fácil la lectura en el intérprete, que no se muestran en está para simplificar una primera lectura del mismo.

2 Un signo = significa asignación mientras que dos (==) significa igualdad, ver manual.

3 en el curso de 1°CB, alcanza con que el resto de la división es 0, o sea división exacta)

## Actividad 2:

*El siguiente es un programa elaborado en python.*

```
a = input("escriba un número natural > 0 o 0 para finalizar ")
while a > 0:
        c= input ("escriba una cota ")
        m = [a*x for x in range (c//a+1)]
        print "los multiplos de ", a, "hasta ", c, "son ", m
        a = input("escriba un número natural > 0 o 0 para finalizar ")
print "Gracias por usar mi programa!"
```

Responde las siguientes cuestiones:

a) Ejecuta el programa en forma manual (sin usar la computadora), para a= 5 y c=60, registrando todos los pasos. Realiza lo mismo para a=7 y c=60
b) Prueba con otros valores de a y c.
c) ¿Para qué sirve el programa?
d) ¿Cuál es la especificación de entrada? ¿Qué información devuelve el programa?.
e) Escribe los pasos del programa en español.
f) Copia el programa en gedit y ejecútalo en phyton, ¿funciona como imaginabas?

### ¡¡¡ Ayuda !!!

Los comandos en los lenguajes de programación se escriben en inglés, te recordamos el significado de algunas de las palabras y comandos que aparecen:

input: introducir, ingresar, registrar.
While: mientras
print: imprimir, escribir
*for:* para
*in:* en

*división entera:* el cociente de la división entera se obtiene con el operador // y el resto con el operador %. Por ejemplo: 10//4 es 2 y 10%4 es 2
La función range en python:

***range(1,a+1)***, genera una lista de los naturales del 1 al a. Por ejemplo range(1,6) genera la lista de los naturales entre 1 y 5, [1,2,3,4,5].

***range (n):*** devuelve la lista de los naturales entre 0 y n-1. Por ejemplo range (4) = [0,1,2,3]

## Los objetivos de está segunda actividad son:

• Desde el punto de vista de los conceptos matemáticos, trabajar la dualidad entre las nociones de divisor y múltiplo[1]. Se podrá formalizar alguna definición de múltiplo y conjunto de múltiplos.

• Desde el punto de vista de la programación seguir familiarizando al estudiante con el lenguaje python al realizar traducciones que permitan interpretar los comandos y la sintaxis del lenguaje. Se agrega el uso de algunos comandos y operadores, que están explicados en la ayuda.

• Finalmente en la integración de las dos disciplinas, hacer evidente los diferentes niveles de abstracción entre un manejo puramente matemático de los conceptos y un manejo que implique un procedimiento algorítmico.

En el primer caso puedo imaginar y representar el conjunto de múltiplos[2] de a como m(a)={0,a,2a, ,ia, .....}, o m(a)={n.a/n∈N} donde representar e imaginar un conjunto infinito no parece ser un problema.

En el caso del procedimiento algorítmico, interviene el factor tiempo que inhabilita la posibilidad de realizar un procedimiento infinito por lo cual es necesario usar una cota.

* Un ejemplo de respuesta esperada para el item a) podría ser:

| | |
|---|---|
| a = input("escriba un número natural > 0 o 0 para finalizar ") | 5 |
| while a > 0:<br>c= input ("escriba una cota ") | 60 |
| m = [a*x for x in range (c//a+1)] | podemos poner range (60//5 + 1) que como ya se explicó es [0, 1, 2, ..., 12]<br>m es la lista de los elementos a *x tomando los x de [0, 1, 2, ..., 12], o sea<br>m={5*0,5*1,5*2, 5*12} |
| print "los múltiplos de ", a, "hasta ", c, "son ", m | Se ve en la pantalla: los múltiplos de 5 hasta 60 son [0,5,15,20,25,30,35,40,45,50,55,60] |
| a = input("escriba un número natural > 0 o 0 para finalizar ") | 7<br>como 7 es distinto de 0 pide una cota y vuelve a calcular |
| c= input ("escriba una cota ") | 60 |
| m = [a*x for x in range (c//a+1)] | range (60//7 + 1) es [0,1,2, ..., 8]<br>por lo tanto<br>m={7*0,7*1,7*2, 7*9} |

1a*b=c implica que a y b (si son no nulos) son divisores de c y que c es múltiplo de a y de b.

| | |
|---|---|
| print "los multiplos de ", a, "hasta ", c, "son ", m | Se ve en la pantalla: Los múltiplos de 7 hasta 60 son: [0,7,14,21,28,35,42,49,56] |
| a = input("escriba un número natural > 0 o 0 para finalizar ") | 0 |
| print "Gracias por usar mi programa!" | Se ve en la pantalla: Gracias por usar mi programa! |

Está implementación incluye varios comandos nuevos, sin embargo ofrece la oportunidad de trabajar en otras competencias, como la lectura y la búsqueda de información en un texto: la parte Ayuda!!! del enunciado del problema. Seguramente la actividad resultará más enriquecedora si los estudiantes la resuelven en pequeños grupos.

Por otro lado, en el momento de ejecutar el programa, se puede analizar, cuáles son los comandos que permiten interactuar y cuáles son "internos" del programa.

* Posible actividad de profundización: Escribe un algoritmo que dado un natural te permita hallar sus h primeros múltiplos.

### Actividad 3:

*Diseña un algoritmo que te permita hallar el máximo común divisor a dos naturales a y b dados.*

- *Se deben seguir las siguientes etapas:*
- *Indicar los datos a ingresar y la información que debe devolver el programa.*
- *Escribir el procedimiento en español, probarlo para valores concretos de a y b.*
- *Escribir el procedimiento en phyton, realizar las pruebas y ajustes necesarios.*

Está es la primera actividad en la que los estudiantes deberán proponer una solución completa al problema. Los algoritmos propuestos dependerán de los conocimientos matemáticos de cada grupo de estudiantes (intersección de conjuntos, algoritmo de Euclides, etc).

* La especificación del problema es:

**# Especificación de entrada:** Un par de números naturales, a y b
**# Especificación de elementos de salida:** un número natural que es el máximo común divisor de a y b (MCD(a,b))

Desarrollaremos en primer lugar el algoritmo que utiliza intersección de conjuntos, que nos permitirá introducir el uso de la idea de función, y luego el del Algoritmo de Euclides, que nos proporcionará un ejemplo de función definida recursivamente.

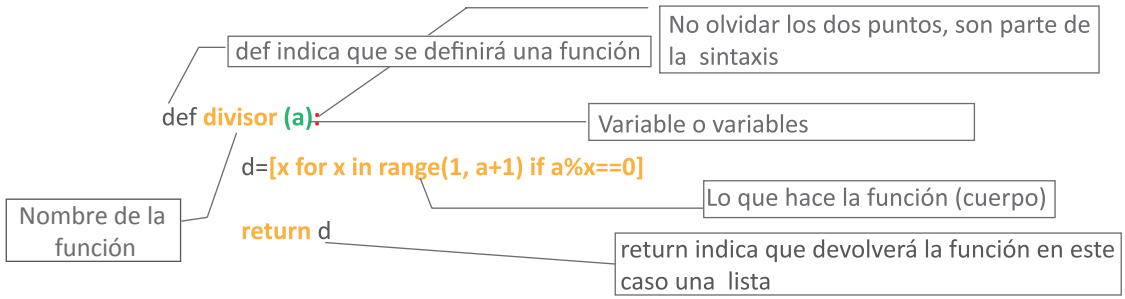
* Funciones

La sintaxis general de la definición de una función en python es[1]:

```
def nombre (lista de variables):
        cuerpo
        return resultado
```

Para la función divisores[2]:

```
def divisores(a):
        d = [x for x in range(1, a+1) if a%x==0]
        return d
```




### 1 - Utilizando intersección de conjuntos:

Se cuenta con el programa que permite hallar los divisores de un número que puede utilizarse como una función.

Siguiendo la definición de MDC(a,b) que dice que es el máximo de los divisores comunes a a y a b, tenemos un posible algoritmo:

1. Ingrese un par de números naturales (a,b)
2. Hallar d(a) (conjunto de divisores de a)
3. Hallar d(b) (Conjunto de divisores de b)
4. Hallar d= d(a) $\cap$d(b) (d es el conjunto de los divisores comunes)
5. MCD(a,b)=D es el máximo de d.
6. Escriba D.

Ya tenemos definida arriba la función para hallar el conjunto de divisores de un número (items 2 y 3). Para el item 4 es necesario hallar la intersección de dos listas y para el ítem 5 necesitamos hallar el máximo de un conjunto ordenado (una lista).

Crearemos un archivo llamado divis donde definiremos las funciones necesarias para implementar el algoritmo:

1. divisores: que dado un número nos da el conjunto de sus divisores.

---

1 en python para definir funciones no es necesario determinar dominio y codominio, en otros lenguajes debemos explicitarlos.

2 Ver archivo divis.py

2. intersección: que devuelve la intersección de dos conjuntos dados, representados por listas.
3. divisores comunes: que devuelve el conjunto de elementos comunes a dos conjuntos dados.

Luego iremos agregando las funciones que se vayan creando y necesitando en los siguientes problemas.

Debemos tener en cuenta que las funciones no leen datos ni imprimen resultados, sólo calculan. Por lo tanto tomaremos del algoritmo que implementamos para calcular el conjunto de divisores, solo la parte en la que se hace el cálculo, dejando el ingreso de datos y la impresión del resultado en pantalla, para otro programa.

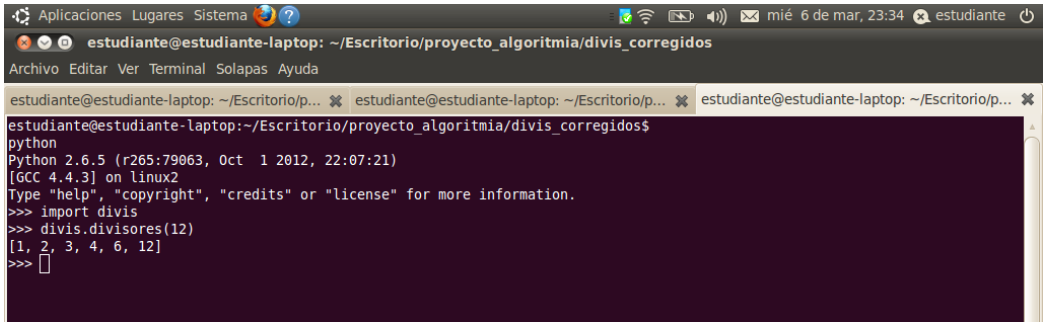
Otra cosa que no se puede olvidar de la sintaxis es la indentación, el return tiene más sangría que el def. (ver errores frecuentes en manual).

Pasaremos a mostrar las funciones que corresponden a los siguientes pasos, definidas todas en un mismo archivo de texto: divis.py:

| | |
|---|---|
| def divisores(a):<br>d = [x for x in range(1, a+1) if a%x==0]<br>return d | Ya fue comentado |
| def inter(l1,l2):<br>l = []<br>for element in l1:<br>if element in l2:<br>l.append(element)<br>return l | Está función tiene como variable (en programación se suele usar el término:parámetro o argumento de la función.)<br>un par de listas (l1 y l2), y devuelve una lista l con los elementos que pertenecen a ambas (la intersección) |
| def divisorescom(a,b):<br>d1 = divisores(a)<br>d2 = divisores(b)<br>dc = inter(d1,d2)<br>return dc | Está función tiene como variable un par de naturales (a y b), halla los divisores de cada uno usando la función divisores, y devuelve en dc la intersección de los dos conjuntos usando la función inter |
| def MCD(a,b):<br>dc = divisorescom(a,b)<br>D = max(dc)<br>return D | Está función tiene como variable un par de naturales(a y b), halla el máximo del conjunto de divisores comunes a los números a y b y lo devuelve en la variable D. |

La implementación en el intérprete de la función MCD se muestra en el archivo maxcdiv.py.
Explicaremos ahora cómo usar las funciones definidas en divis en el intérprete.
Para usar la función en el intérprete, debemos importar el módulo (archivo) divis y luego aplicar la función escribiendo divis.divisores(a), siendo a un valor concreto, por ejemplo para halar los divisores de 12:

```
>>> import divis
>>> divis.divisores(12)
[1, 2, 3, 4, 6, 12]
>>>
```




Otra opción es crear un módulo para que el intérprete aplique la función, aparecen entonces los comandos que interaccionan con el usuario del programa.

A modo de ejemplo, la siguiente sería la implementación del módulo que permite hallar el conjunto de divisores de un número usando la función divisores definida en divis. (Archivo divisores.py, es un archivo de texto)

```
import divis

a = input("escriba un número natural mayor que 0 o 0 para finalizar: ")
while a > 0:
        d = divis.divisores(a)
        print "los divisores de ", a, "son", d
        a = input("escriba un número natural mayor que 0 o 0 para finalizar: ")
print "Gracias por usar mi programa!"
```

Para usar este módulo debemos escribir en el intérprete

```
>>> import divis
>>> import divisores
```

Mostramos ahora como se "ve" en el intérprete.

```
Type "help", "copyright", "credits" or "license" for more information.
>>> import divis
>>> import divisores
escriba un numero natural mayor que 0 o 0 para finalizar: 12

los divisores de  12 son [1, 2, 3, 4, 6, 12]

escriba un numero natural mayor que 0 o 0 para finalizar: 25

los divisores de  25 son [1, 5, 25]

escriba un numero natural mayor que 0 o 0 para finalizar: 37

los divisores de  37 son [1, 37]

escriba un numero natural mayor que 0 o 0 para finalizar: 0
Gracias por usar mi programa!
>>>
```

2 - Utilizando el algoritmo de Euclides:

En el curso de 2do de bachillerato, se introduce el algoritmo de Euclides como uno de los mecanismos para hallar el MCD de dos naturales.

La propuesta de diseñar un módulo que resuelva el problema usando el algoritmo de Euclides, permite no solo introducir la noción de funciones definidas recursivamente, si no de reflexionar en la propiedad sobre la que se construye el algoritmo.

Cuando se define una función recursivamente aplicamos la misma función en cada paso a variables que van disminuyendo (caso inductivo) hasta llegar a un caso conocido (caso base). Tenemos que identificar entonces que es lo que hacemos de la misma manera en cada paso.

* Algoritmo en español:

1. Dados dos números a y b
2. realiza la división de a entre b cuyo resto es r1
3. si r1 es 0 entonces MCD(a,b)=b
4. si no se realiza la división de b entre r1 cuyo resto es r2
5. si r2 es 0 entonces MCD(a,b)=r1
6. si no se repite el procedimiento hasta obtener resto 0.
7. MCD (a,b) =r(n-1) (el resto anterior al nulo)

Sin embargo en está descripción del algoritmo de Euclides, no se evidencia la recursión, se muestra la sucesión de pasos, pero no está claramente identificada la propiedad que valida el procedimiento.

La clave está en la siguiente propiedad:

MCD(a,b)=MCD(b,r1)=MCD(r1,r2)=.....MCD(r(n-1),rn)

y el hecho de que en un número finito de pasos se alcanza resto 0.

Una mejor descripción del algoritmo, con el objetivo de definir una función recursiva es el siguiente:

| | |
|---|---|
| Sea MCD(a,b) | El algoritmo se llama MCD y se aplica a (a,b) |
| Sea r es el resto de la división de a entre b . | Defino r |
| si r es 0 entonces devuelvo b | Si r es 0 finalizo, ya tengo el resultado |
| si no MCD(b,r) | Si no, vuelvo a hacer lo mismo con b y r, es decir, aplico el algoritmo MCD al par (b,r) |

* La función en matemática:

$$mcd : N\,x\,N \rightarrow N$$

$$mcd(a,b) = \begin{cases} b & si\,(a \bmod b) = 0 \\ mcd(b, a \bmod b) & si\,(a \bmod b) \neq 0 \end{cases}$$

Utilizando python la función se implementa:

```
def MCDr(a,b):
    r=a%b
    if r==0:
        return b
    else:
        return MCDr(b,r)
```

La implementación en el intérprete, se ve en el archivo: euclides.py

## Actividades de ampliación y consolidación:

1. ¿Cómo saber si un número es primo?
   1.1 Crea una función que te permita decidir si un número es primo o no.[1]
   1.2 Implementa en el intérprete de python dicha función.[2]
2. ¿Cuáles son los números menores que 100 con únicamente un divisor impar? ¿Y los menores de 500?
3. Dado un número n, ¿Cuáles son los números menores o iguales a n que al dividirlos entre 5 dan resto 3?
3. Clasifica los números menores que un n dado según los restos de dividirlos entre 5.

1 ver solución en divis.py

2 ver solución en esprimo.py

## Conclusiones y reflexiones:

Con esta propuesta se busca integrar los principios y métodos de la algoritmia a la clase de matemática. Creemos que lejos de suponer un esfuerzo extra, brinda al profesor la posibilidad de renovar sus estrategias de trabajo en clase, favoreciendo un mayor involucramiento de los estudiantes que se enfrentan a desafíos novedosos.

En lo que refiere a los contenidos matemáticos, mejora su comprensión, permitiendo una nueva forma de trabajar con ellos, por ejemplo en lo que refiere a funciones. También permite trabajar con la estructura de la matemática en lo que refiere a las definiciones , al hacer evidente la necesidad de elegir la definición adecuada al algoritmo de resolución que está condicionado a las herramientas que ofrece el lenguaje.

Sin embargo, creemos que el mayor aporte de esta integración refiere al desarrollo de algunas competencias como: resolución de problemas, comunicación y representaciones [3].

En resolución de problemas permite reforzar la comprensión del problema al ser estrictamente necesario explicitar cuáles son los datos y qué se busca (especificación del problema).

Además ofrece oportunidades de analizar como un mismo problema puede ser resuelto por diferentes algoritmos o procedimientos. Por otro lado, se favorece el desarrollo de la abstracción, cuando se descompone un problema en sub-problemas que mejoran su comprensión y resolución. Este aspecto fue ejemplificado en la última actividad, donde se hizo evidente la ventaja del trabajo con funciones.

En lo que refiere a las competencias comunicativas y de representación, las actividades propuestas ofrecen instancias tanto de codificación como de decodificación, así como de experimentar la dualidad texto - acción de los programas de computación [1].

Finalmente creemos que se fortalece el desarrollo de la autonomía y construcción del rol de estudiante, ya que la falta de rigor impide que los programas funcionen, dejando al docente afuera de la decisión de si la respuesta es o no correcta [1].

Como reflexión sobre la experiencia queremos destacar dos aspectos principales.

El primero que no resultó sencillo como docentes de matemática, sin formación en programación, entrar en la lógica y el lenguaje del programa python. Como todo contenido nuevo, nos generó varios conflictos a resolver. Sin embargo fueron estos los que nos permitieron comprender el aporte que brinda este tipo de trabajos para nuestra tarea como docentes.

El segundo es que existen muchos conceptos, que en apariencia son el mismo, pero que cada disciplina construye sus propias imágenes mentales al respecto. Esto nos conduce a la reflexión de lo variadas que pueden llegar a ser las imágenes mentales[1] de nuestros alumnos y la importancia de ofrecer un abanico variado de experiencias que permita una mejor com-

1 'Imagen conceptual' refiere a 'algo' no verbal asociado en nuestra mente con el nombre del concepto. (“The concept image is something non-verbal associated in our mind with de concept name” ) [4].

prensión de los conceptos. Este aspecto es uno de los aportes más importantes del trabajo interdisciplinario entre docentes.

Queda pendiente la aplicación de las actividades en clase, lo que seguramente nos permitirá realizar los ajustes necesarios para un mejor aprovechamiento, cumpliendo con el ciclo: especificación del problema, diseño de una solución , implementación y ajustes.

## Bibliografía:

1. Gilles Dowek, Quelle informatique enseigner au lycee?, https://who.rocq.inria.fr/Gilles.Dowek/lycee.html (2005)

2. D. Harel, Algorithmics: The Spirit of Computing, Addison-Wesley, Reading, MA, (425 pp.) 1987. 2nd edition, 1992; 3rd edition, 2004.

3. NCTM. , 2003 PRINCIPIOS Y ESTÁNDARES PARA LA EDUCACIÓN MATEMÁTICA. Traducción de Manuel Fernández SAEM Thales Sevilla 2003 ISBN 84-933040-3-4

4. Vinner, S. (1991): “The role of definitions in the teaching and learning mathematics” en Advanced Mathematical Thinking”. Ed David Tall, Kluwer, London.

# GEOMETRÍA ANALÍTICA

Daniela Pagés, Teresita Carrión, CFE, Montevideo

Hemos decidido elaborar una secuencia de Geometría Analítica, porque es un tema con mucho contenido algorítmico. Pensamos que los estudiantes, al elaborar los algoritmos de resolución de los distintos problemas, están desarrollando el pensamiento computacional y recuperan con el uso, las herramientas que han ido trabajando antes y en el propio curso.

Estas actividades están pensadas para un curso de Matemática de Segundo Año de Bachillerato de cualquier diversificación.
En la unidad de Geometría Analítica de ese curso aparecen los siguientes temas:

- Coordenadas cartesianas en el plano.
- Ecuación cartesiana de la recta. Semiplano.
- Distancia entre dos puntos.
- Ecuación de la circunferencia. Círculo.
- Intersección de recta y circunferencia.
- Resolución gráfica de sistemas de inecuaciones lineales en el plano.

En cada actividad, se propondrá a los estudiantes un problema para resolver "a mano" en un primer momento, y luego se les pedirá a ellos que formulen el problema a la "máquina", que identifiquen los datos de entrada y de salida, que escriban el algoritmo usando lenguaje coloquial y por último que implementen el algoritmo usando el lenguaje elegido: python.

Se comienza trabajando con actividades que permitan valorar la situación de los estudiantes respecto a: sus conocimientos acerca del sistema cartesiano de coordenadas, la representación de los puntos en un sistema cartesiano, la determinación de coordenadas (en especial, la resignificación de los puntos de los ejes como dotados de dos coordenadas). Transversalmente se trabajarán los números reales.

* Los objetivos de esta secuencia son que los estudiantes:

- Recuperen los conceptos y herramientas trabajados en los cursos anteriores (sistema cartesiano, representación de puntos, ecuación de una recta).
- Consoliden la relación entre la pertenencia de un punto a una figura de la que se co-

noce la ecuación, y la relación algebraica de las coordenadas del punto con la ecuación.

• Resuelvan problemas que les permitan acercarse a los contenidos de la unidad, en la conjunción del trabajo individual y colectivo.

• Planteen sus propios algoritmos de resolución de los problemas planteados.

## Actividad 1

a) Representa en un sistema cartesiano ortogonal los puntos A = (2, 3); B = (-5, -3); C = (1, -4); P = (0, -2); Q = (1, 0), el simétrico de cada uno respecto al eje Oy, respecto al eje Ox, y respecto al origen de coordenadas.

b) Dado un punto por sus coordenadas, determina qué coordenadas tendrá el punto simétrico del mismo con respecto al eje de las abscisas. Explica con palabras cómo modificarías las coordenadas del punto dado para obtener las coordenadas de su simétrico respecto al eje de las abscisas, y luego exprésalo simbólicamente.

c) Idem respecto al eje de ordenadas.

d) Idem respecto al origen.

## Actividad 2

Representa en un sistema cartesiano los puntos A = (2, 3), B = (-1, 4), C = (-3, 2).
Calcula el perímetro del triángulo ABC.

A partir de este ejercicio se discutirá con los estudiantes la posibilidad de encontrar una expresión para el cálculo de la distancia. Se planteará esto como tarea para realizar en parejas o pequeños grupos. Esto constituye la siguiente actividad.

## Actividad 3

Imagina que tienes que calcular el perímetro de un octógono, conociendo las coordenadas de todos sus vértices. Será necesario calcular las medidas de todos los lados.
¿Puedes escribir un método para calcular la distancia entre dos puntos A = (a, b) y B = (c, d)? Escribe todos los pasos y acompaña lo que escribiste con una representación.

Se discutirá colectivamente a partir de lo que han hecho los estudiantes. Se espera que algunos hayan utilizado el teorema de Pitágoras como herramienta. Puede resultarles difícil, sin embargo, encontrar una expresión general. Una vez que se haya llegado a plantear una expresión para la distancia, se preguntará (si no ha surgido en algún grupo), qué sucede si los dos puntos tienen igual abscisa o igual ordenada. Se discutirá la validez de la expresión en estos casos.
A partir de esto se planteará a los estudiantes que en el transcurso de esta unidad, y otras del curso, muchas veces tendremos que calcular la distancia entre dos puntos. Y que

además de este cálculo, habrá algunas rutinas que se repetirán en muchos ejercicios. Se planteará entonces la conveniencia de desarrollar un algoritmo con los pasos para calcular, por ejemplo, la distancia entre dos puntos, usando un lenguaje de programación que nos permita introducir los datos y que la máquina resuelva el problema. Se les planteará que, si realizan el programa, luego podrán usarlo cada vez que lo necesiten.

Se preguntará a partir de lo anterior: ¿qué datos debemos ingresar en el programa, y qué resultado obtenemos? A partir de las respuestas se enunciará el problema:

* Especificación del problema:

Dados dos puntos A y B por sus coordenadas A= (a, b), B= (c, d), determinar la distancia m entre ellos.

* Especificación de la entrada:

Dos pares ordenados de reales (a, b) y (c, d).

* Especificación de elementos de salida:

El valor de la distancia m.

Luego pasamos a escribir una solución del problema, o sea un algoritmo, usando primero lenguaje coloquial, es decir, el algoritmo en español y después la implementación en python:

| ***Algoritmo en español*** | ***Implementación en python*** |
|---|---|
| | import math * |
| (a,b) son las coordenadas del punto A | (a,b) = input ("ingrese las coordenadas de A:") |
| (c,d) son las coordenadas del punto B | (c,d) = input ("ingrese las coordenadas de B:") |
| Calcular d = sqrt((a – c)2 + (b – d)2) | m = math.sqrt ((a – c)**2 + (b – d)**2) |
| Escribe "la distancia entre A y B es m" | print "la distancia entre A y B es ", m |

Previamente a la implementación en python, se les explicará a los estudiantes sobre el concepto de implementación, se presentará python, los principales comandos del lenguaje (se les entregará un manual sencillo de uso, que puedan consultar) y se implementará este algoritmo, propiciando el trabajo colectivo.

Una vez implementado el algoritmo, se pedirá a los estudiantes que lo utilicen rehaciendo la actividad 2 o el cálculo de un polígono de mayor número de lados.

## Actividad 4

Escribe e implementa un algoritmo que te permita, a partir de las coordenadas de un punto, determinar las coordenadas del simétrico de dicho punto con respecto al eje de abscisas. Lo mismo para el eje de ordenadas, y el origen. (Actividad 1).

Esta actividad está pensada para que los estudiantes practiquen, más que nada, el uso del lenguaje, en un ejemplo muy sencillo.

Se repite la idea de plantear la especificación del problema, de la entrada y de la salida y la redacción del algoritmo en español y en python.

* import es el comando que permite usar un módulo , en este caso el módulo math en el que se definen las funciones matemáticas usuales.

* Especificación del problema:

Dado un punto A por sus coordenadas, A = (a, b), determinar las coordenadas del punto simétrico de A con respecto al eje de las abscisas.

* Especificación de la entrada:

Las coordenadas (a, b) de A.

* Especificación de elementos de salida:

El par ordenado (a, -b).

| ***Algoritmo en español*** | ***Implementación en python*** |
|---|---|
| (a,b) son las coordenadas del punto A<br>Obtener (a, - b)<br>Escribe "las coordenadas del simétrico de A respecto al eje de las abscisas son (a, - b)." | (a,b) = input ("ingrese las coordenadas de A:")<br><br>print "las coordenadas del simétrico de A respecto al eje de las abscisas son", (a, - b). |

Antes de la actividad siguiente, se estudiará la ecuación cartesiana de la recta y la inecuación de un semiplano.

## Actividad 5

a) Dado el punto A = (2, 3) y la recta r de ecuación: - 2x + 3y – 5 = 0, determina si el punto A pertenece a la recta r. Justifica tu respuesta.

b) Dado un punto A por sus coordenadas, A= (x0, y0), y una recta r por su ecuación:
ax + by + c = 0, determinar si el punto pertenece o no pertenece a la recta.

Explica en lenguaje coloquial y luego piensa una implementación en python para resolver el problema.

En la parte a) se espera que los estudiantes sustituyan las variables de la ecuación por las coordenadas respectivas de A, y lleguen a la conclusión de que el punto pertenece a la recta. La condición necesaria y suficiente se habrá trabajado al deducir la ecuación de una recta por dos puntos.

En la segunda parte, luego de discutir su resolución, se pedirá a los estudiantes que planteen e implementen un algoritmo para resolver el problema.

* Especificación del problema:

Dado un punto A por sus coordenadas, A= (a,b), y una recta r por su ecuación:
cx + dy + e = 0, determinar si el punto pertenece o no pertenece a la recta.

* Especificación de la entrada:

Las coordenadas (a, b) de A y los coeficientes (c, d, e) de la ecuación de la recta r.

* Especificación de elementos de salida:

Uno de los siguientes mensajes: "El punto A pertenece a la recta r" o "el punto A no pertenece a la recta r".

| Algoritmo en español | Implementación en python |
|---|---|
| (a,b) son las coordenadas del punto A<br>cx + dy + e = 0 es la ecuación de la recta r<br>Calcular p = c*a + d*b + e<br>Si p=0, escribe: “el punto A pertenece a la recta”;<br>Si no, escribe: “el punto A no pertenece a la recta”. | (a,b) = input (“ingrese las coordenadas de A:”)<br>(c,d,e) = input ("ingrese los coeficientes de la recta (r) de ecuación cx+dy+e=0: ")<br>p = a*c + b*d + e<br>if p== 0:<br>print "El punto A pertenece a la recta r"<br>else:<br>print "El punto A no pertenece a la recta r" |

En esta implementación se agregan dos nuevos conceptos de programación: la indentación y las instrucciones de selección .

## Haciéndonos de una lista de fórmulas

Llegados a este punto, se propondrá a los estudiantes hacerse su propia “hoja” de fórmulas, pero en forma de algoritmo, para ser ejecutado por la computadora en python. Y que esas fórmulas elaboradas por ellos, podrán usarlas en cualquier momento del curso, inclusive en las evaluaciones.

Aquí se discutirá con ellos la posibilidad de elaborar un módulo con todas las definiciones que luego se utilizarán en otros algoritmos, de forma similar al módulo math de python.

Se podrá incluir en él la definición de distancia entre dos puntos, distancia entre un punto y una recta, averiguar si un punto pertenece a una recta, y algunos otros que pudieran sugerir los estudiantes.

* Implementación del módulo geo.py

```
# Formulas de geometria analitica
# distancia entre dos puntos
import math
def distancia ((a,b), (c, d)):
    m = math.sqrt((a - c)**2 + (b - d)**2)
    return p
# determinante de una matriz 2x2
def det ((a,b),(d,e)):
    m = a*e-b*d
    return
# sustituye las coordenadas (a,b) de un punto en la ecuacion de una recta cx+dy+e=0
def eva ((a,b),(c,d,e)):
    p = a*c + b*d + e
    return p
```

## Mejorando las implementaciones

Los estudiantes podrán comprobar una vez que empiecen a utilizar los programas, que cada vez que quieran a volver a usar uno de ellos, deben ejecutarlo desde el principio. Para ahorrar esfuerzos, le sugeriremos utilizar la instrucción de iteración “while”, para que pue-

dan repetir el mismo algoritmo todas las veces que lo deseen.

La sintaxis es:

```
s = input ("Si desea finalizar, ingrese 0, si va a continuar, ingrese 1: ")        (al inicio)
while  s == 1:
(indentacion) # aquí va el cuerpo del programa
                    s = input ("ingrese 0 para finalizar o 1 para continuar: ")       (al final)
```

También podemos agregar un mensaje al principio, enunciando la función del programa y otro al final de despedida.

Ejemplos:

```
print " Este programa calcula la distancia entre dos puntos a partir de sus coordenadas, todas las veces que necesites hacerlo"
print "Gracias por usar mi programa"
```

Incorporando estas mejoras, la implementación de la actividad 3, quedaría de la siguiente manera:

```
import math
import geo
print "Este programa calcula la distancia entre dos puntos a partir de sus coordenadas, todas las veces que necesites hacerlo"
s = input ("Si desea finalizar, ingrese 0, si va a continuar, ingrese 1: ")
while s == 1:
        (a,b) = input ("ingresa las coordenadas del punto A: ")
        (c,d) = input ("ingresa las coordenadas del punto B: ")
        m = geo.distancia  ((a,b),(c,d))
    print "La distancia entre A y B es ", m
    s = input ("ingrese 0 para finalizar o 1 para continuar: ")
print "Gracias por usar mi programa"
```

## Actividad 6

a) Investigar si las rectas r y s son secantes o paralelas, para cada uno de los siguientes casos. Para cada par de rectas secantes, determina las coordenadas de su punto de intersección.

i) r) $x + 3y - 5 = 0$ s) $2x + 6y + 1 = 0$

ii) r) $y = 4x - 1$ s) $2x + y - 5 = 0$

iii) r) $-2x + y - 4 = 0$ s) $4x - 2y + 8 = 0$

b) Dadas dos rectas por sus ecuaciones: r) $ax + by + c = 0$ y s) $mx + ny + p = 0$, escribe e implementa un algoritmo para determinar las coordenadas del punto P de intersección de r y s, cuando existe.

Con la primera parte del ejercicio se busca que los estudiantes investiguen condiciones analíticas para las posiciones relativas de dos rectas.

* Especificación del problema:

Dadas las ecuaciones de dos rectas, determinar si son paralelas o secantes, determinando, en este caso, las coordenadas de su punto de intersección.

* Especificación de la entrada:

Los coeficientes de las ecuaciones de dos rectas.

* Especificación de elementos de salida:

Uno de los siguientes mensajes: "Las rectas son coincidentes" o "las rectas son paralelas disjuntas" o el par "Las rectas se cortan en el punto", coordenadas del punto de corte.

* Algoritmo en español:

ax + by + c = 0 es la ecuación de la recta r1
dx + ey + f = 0 es la ecuación de la recta r2
calcular m = a*e – d*b
calcular n = c*e – f*b
calcular p = a*f – d*c
Si m es distinto a 0
Calcular xP = n/m
Calcular yP=p/m
escribir "las rectas se cortan en el punto (xP,yP)
o si n=0
escribir "las rectas son coincidentes
o si no
escribir "las rectas son paralelas disjuntas"

* Implementación en python

```
import math
import geo
print "Este programa discute la posición relativa de dos rectas y calcula el punto de interseccion en caso de existir, todas las veces que sea necesario."
s = input ("Si desea finalizar, ingrese 0, si va a continuar, ingrese 1: ")
while s == 1:
    (a,b,c) = input ("ingrese los coeficientes (a,b,c) de la recta r1: ax+by=c: ")
    (d,e,f) = input ("ingrese los coeficientes (d,e,f) de la recta r2: dx+ey=f: ")
        m = geo.det  ((a,d),(b,e))
    n = geo.det ((c,f),(b,e))
    p = geo.det ((a,d),(c,f))
        if m!=0:
        xP=n/m
        yP=p/m
        print  "Las rectas se cortan en el punto", (xP,yP)
    elif n == 0:
        print "las rectas son coincidentes"
          else:
        print  "las rectas son paralelas disjuntas"
        s = input ("ingrese 0 para finalizar o 1 para continuar: ")
print "Gracias por usar mi programa"
```

## Actividad 7

Escribe un algoritmo que te permita, dada una recta por su ecuación, y dos puntos por sus coordenadas respectivas, determinar si los puntos pertenecen al mismo semiplano de borde la recta, o a distintos semiplanos de borde la recta.

* Especificación del problema

Dadas las coordenadas de dos puntos, y la ecuación de una recta, determinar si los dos puntos pertenecen al mismo o a distintos semiplanos de borde la recta.

* Especificación de la entrada:

Los coeficientes de la ecuación de una recta y las coordenadas de dos puntos.

* Especificación de elementos de salida:

Uno de los siguientes mensajes "Los puntos pertenecen al mismo semiplano de borde r" o "los puntos no pertenecen al mismo semiplano de borde r".

* Algoritmo

(a, b) son las coordenadas del punto A
(c, d) son las coordenadas del punto B
(e, f, g) son los coeficientes de la ecuación de la recta r (ex + fy + g = 0)
Calcular m = a*e + b*f + g
Calcular n = c*e + d*f + g
Si m*n < 0, devolver el mensaje: "los puntos A y B pertenecen a distintos semiplanos de borde r"
Si no, devolver el mensaje: "los puntos A y B pertenecen al mismo semiplano de borde r".

* Implementación

```
import math
import geo
print "Este programa determina si dos puntos dados pertenecen o no al mismo semiplano de borde una recta dada, todas las veces que se lo pidas."
s = input ("Si desea finalizar, ingrese 0, si va a continuar, ingrese 1: ")
while s == 1:
    (a,b) = input ("ingrese las coordenadas (a,b) del punto A: ")
    (c,d) = input ("ingrese las coordenadas (c,d) del punto B: ")
    (e,f,g) = input ("ingrese los coeficientes (e,f,g) de la recta r: ex+fy+g=0: ")
    m = geo.eva  ((a,b),(e,f,g))
    n = geo.eva ((c,d),(e,f,g))
    if m*n < 0:
        print  "Los puntos A y B pertenecen a distintos semiplanos de borde r"
    else:
        print  "los puntos A y B pertenecen al mismo semiplano de borde r"
      s = input ("ingrese 0 para finalizar o 1 para continuar: ")
print "Gracias por usar mi programa"
```

## Actividad 8
## Aplicando lo que hemos aprendido

Sea ABCD un cuadrilátero cualquiera. Por propiedades geométricas se cumple que el cuadrilátero determinado por los puntos medios de sus lados es siempre un paralelogramo.
Diseña un algoritmo y su posterior implementación en python, para comprobar esta propiedad en todos los casos que el usuario del programa lo desee.

Previamente responde estas preguntas:
1) ¿Cuáles serán los datos de entrada?
2) ¿Cuál es la salida?
3) ¿Qué camino tenemos que seguir para que, a partir de los datos se llegue a la salida?

Una posible resolución del problema es la siguiente:

* Especificación de los datos de entrada

Coordenadas de los cuatro vértices de un cuadrilátero ABCD.

* Especificación de la salida

(Dado que lo que se quiere es comprobar una propiedad, la salida será siempre la misma, aunque se debe pensar también en la posibilidad de que no se cumpla).
Uno de los mensajes "El cuadrilátero MNPQ es un paralelogramo, siendo M, N, P y Q los puntos medios de AB, BC, CD, DA respectivamente" o "El cuadrilátero MNPQ no es un paralelogramo, siendo M, N, P y Q los puntos medios de AB, BC, CD, DA respectivamente"

* Algoritmo en español

(a,b), (c,d), (e,f) y (g,h) son las coordenadas de los vértices de un cuadrilátero.
Hallar las coordenadas de M, N, P y Q, puntos medios de los segmentos AB, BC, CD y DA respectivamente.

Escribir las ecuaciones de las rectas MN, NP, PQ y QM
¿MN y PQ son paralelas?
¿NP y QM son paralelas?
Si las respuestas a las 2 preguntas anteriores es sí, el cuadrilátero MNPQ es un paralelogramo.
Si no, el cuadrilátero MNPQ no es un paralelogramo.

* Implementación en python

Previamente se agregan las siguientes definiciones al módulo geo:

```
# coordenadas del punto medio
def pmedio ((a,b),(c,d)):
      m = (a+c)/2.0
   n = (b+d)/2.0
   return (m, n)
# coeficientes de la ecuacion general de la recta determinada por dos puntos
```

```
def recta ((a,b),(c,d)):
   m = b-d
   n = c-a
   p = a*(d-b)-b*(c-a)
   return (m, n, p)
# evalua si dos rectas son paralelas o secantes
def paralelas ((a,b,c),(d,e,f)):

    if (a*e==b*d):
              n = 1
       return n
    else:
       n = 0
       return n
```

## Reflexión final

Si bien esta secuencia ha sido elaborada para todos los cursos de Segundo de Bachillerato, pensamos que probablemente se involucren más con ella los estudiantes de 2º Diversificación Científica. También creemos que puede resultarles difícil aprender a usar el lenguaje python, ya que no tienen cursos de programación. Esperamos, sin embargo, que este trabajo pueda resultarles de utilidad a los docentes, como una forma diferente de abordar la geometría analítica, integrándola con la programación, aunque somos conscientes que su aplicación requiere que éstos reciban una formación básica en programación.

# CONECTIVOS LÓGICOS

* Algoritmos para solucionar problemas sobre fórmulas de la lógica proposicional

Franco Vuan (Escuela Técnica Superior "Pedro Blanes Viale" Mercedes – Soriano)

## OBJETIVO:

Instrumentar una herramienta diferente a la utilizada generalmente en los cursos de Lógica para Computación, con la que el alumno sea capaz de aplicar la lógica para generar algoritmos que contribuyan a la comprensión de las tablas de verdad.

## INTRODUCCIÓN:

Dentro del curso EMT (Educación Media Tecnológica) de Informática en el CETP (Consejo de Educación Técnico Profesional), una de las asignaturas del primer año es Lógica para Computación, la cual es de vital importancia a la hora de la realización de un programa para computadoras, por ejemplo.

Para estos alumnos, el tema del cálculo proposicional es completamente nuevo. Si bien en el lenguaje natural lo utilizan, formalizar el lenguaje e internalizar las operaciones de dicho cálculo se hace a veces algo complejo. Este trabajo intenta contribuir a facilitar la comprensión del tema.

## JUSTIFICACIÓN:

Una de las temáticas del curso es la construcción de tablas de verdad, para describir la forma de operar de los conectores básicos. Estos son:

- "Y" (conjunción)
- "O" (disyunción)
- "ENTONCES" (implica)
- "SI SOLO SI" (equivalencia)

A los alumnos se les dificulta internalizar las diferentes combinaciones de los operadores a la hora de realizar las tablas. Se espera que al crear ellos mismos un pequeño programa, en la que se obtengan los resultados de cada una de las operaciones, estos conceptos quedarán más claros.

## Problema algorítmico

Dados dos valores booleanos (V o F), y un conector lógico, determinar el valor de verdad resultado de aplicar el operador a los valores dados.

Los datos de entrada del problema son
- dos valores: de p que puede ser V o F y q que puede ser V o F
- un conector s, que tiene alguno de los valores "y", "o", entonces, "si solo si"

El resultado (salida) es
- un valor booleano resultado de p s q

En la actividad se usan 0 y 1 para representar V y F (verdadero y falso) y las secuencias de caracteres "Y", "O", "ENTONCES" "SI Y SOLO SI" para representar a cada uno de los conectores lógicos.

## Actividad:

Se hace una introducción previa de los conceptos de proposiciones, simples y compuestas, usando el lenguaje natural. En especial se discute qué es una proposición, y qué no lo es, y que no todo lo que se puede expresar en el lenguaje natural puede ser evaluado como una proposición lógica.

Para ello se toman frases tales como:
- Sale el sol
- Hace calor
- Está despejado

Para luego comenzar a unir esas frases:
- Esta despejado o hace calor

De esta manera se va introduciendo al lenguaje formal
- Sale el sol = p
- Hace calor = q
- Está despejado = r
- Sale el sol y hace calor = p y q
- Sale el sol entonces hace calor = p ENTONCES q

En coordinación con el profesor de Programación, y teniendo en cuenta lo dado por el profesor de Lógica, se pide al estudiante que implemente un programa utilizando el lenguaje Python, para los algoritmos descritos por las tablas de verdad, que se muestran a continuación:

| *p* | *q* | *p y q* | *p o q* | *p entonces q* | *p si solo si q* |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |

Se ingresan los datos y el programa devolverá el resultado de acuerdo al siguiente diseño:

• Se ingresa al programa la operación a realizar representada por las secuencias de caracteres "Y", "O", "ENTONCES", "SI SOLO SI".

• Se ingresa el valor de la primera de las componentes de la operación a la que llamaremos "p" que será un digito de valor 1 o 0, ya que en estas operaciones los operandos tienen valores del álgebra booleana, o sea binarios, 0 o 1.

• Se ingresa el valor del segundo operando que llamamos "q", también representado por 0 o 1.

• Como resultado se obtendrá un binario que dependerá del resultado de la operación ingresada, aplicada a p y a q, siguiendo los pasos del algoritmo representado por la tabla correspondiente.

* Implementación en python de las funciones correspondientes a cada uno de los conectores

```
def funy(p,q):
        if (p == 1) and (q == 1):
                r = 1
                return (r)
        else:
                r = 0
                return (r)
def funo(p,q):
        if (p == 0) and (q == 0):
                r = 0
                return (r)
        else:
                r = 1
                return (r)
def fune(p,q):
        if (p == 1) and (q == 0):
                r = 0
                return (r)
        else:
                r = 1
                return (r)
def funsos(p,q):
        if (p == q):
                r = 1
                return (r)
        else:
                r = 0
```

```
        return (r)
```

* Programa principal

```
import modulos
a = str (raw_input("Ingrese el tipo de operacion logica a realizar  Y; O; ENTONCES; SI SOLO SI, S para Salir del programa: " ))
while a <> "S":
        p = int (input ("Ingrese el valor de p cero o uno: "))
        q = int (input ("Ingrese el valor de q cero o uno: "))
        if (a == 'Y'):
                print "selecciono conectivo Y"
                print "el resultado es :", (modulos.funy (p,q))
        if (a == 'SI SOLO SI'):
                print "selecciono conectivo SI SOLO SI"
                print "el resultado es :", (modulos.funsos (p,q))
        if (a == 'O'):
                print "selecciono conectivo O"
                print "el resultado es :", (modulos.funo (p,q))
        if (a == 'ENTONCES'):
                print "selecciono conectivo ENTONCES"
                print "el resultado es :", (modulos.fune (p,q))
        a = str (raw_input("Ingrese el tipo de operacion logica a realizar  Y; O; ENTONCES; SI SOLO SI, S para Salir del programa: " ))
else :
        print "Gracias por usar mi programa!"
```

# BREVE MANUAL DE PROGRAMACIÓN EN PYTHON

*Agradecemos a Luis Langón por el esfuerzo dedicado en la elaboración de este manual.*

## Contexto

Este manual se elabora en el marco de la edición 2013 del programa de la ANII "Acortando Distancias", y como parte de la pasantía "Algoritmia e integración de programación al proceso de resolución de problemas en cursos de matemática de la enseñanza media. Didáctica de la Informática." Este es un complemento de los capítulos anteriores que contienen los temas trabajados anteriormente. Todos los materiales están disponibles en: www.fing.edu.uy/~darosa/AC2013 y en http://www.anep.edu.uy/prociencia/

## Introducción

Este manual está dirigido a profesores de matemática de enseñanza media. Su objetivo es que cuenten con una herramienta de rápido y fácil acceso en el momento de trabajar con problemas como los planteados en el librillo u otros similares, en las clases de ciclo básico y bachillerato.

Los ejemplos y comentarios que aquí se utilizan fueron hechos presuponiendo que se utiliza un sistema Linux - Ubuntu 10.04 – Gnome, que está generalmente instalado en las computadoras entregadas a profesores y alumnos en el marco del Plan Ceibal.

Trabajaremos con el ejemplo: *"Raíces reales de una ecuación de segundo grado".*

El problema consiste en que dados los coeficientes de una ecuación de segundo grado, se quiere obtener sus raíces reales, o un mensaje "esta función no tiene raíces reales".

## Intérprete python y comandos directos

Al abrir el intérprete python aparece una ventana en la que podemos escribir. La utilizaremos para aprender algunos comandos en lenguaje python (instrucciones que la computadora debe ejecutar).

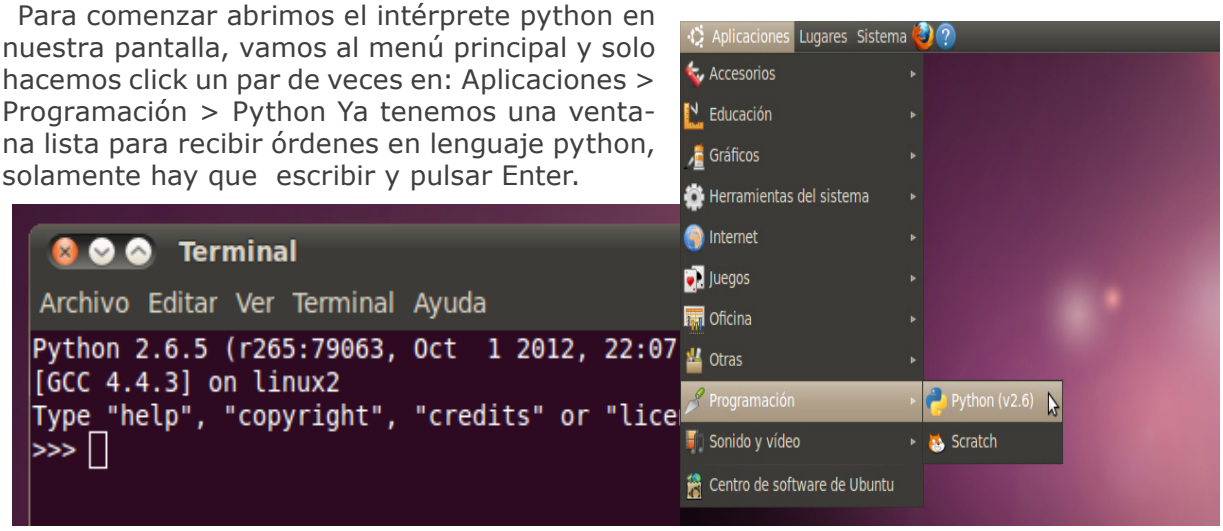
Si queremos que la computadora escriba una frase en la pantalla, por ejemplo, *hola mundo*, entonces le daremos la orden para que lo haga, pero claro que la computadora no entiende español, ni inglés ni ningún otro idioma de los que podemos hablar nosotros, solo entiende cosas como 1000101000100 111010010111010101000101 que nosotros no comprendemos. Por eso necesitamos alguien que haga de intérprete, es decir que podamos escribir en un lenguaje "humano" y el intérprete lo traduzca al lenguaje de la computadora (código de máquina) y ésta pueda ejecutar lo pedido. Dicho "lenguaje humano" es un lenguaje de programación, en este caso python[1]. Debemos conocer las palabras que hay en este lenguaje y cuáles son su sintaxis y su semántica, pero antes diremos algunas palabras acerca de por

1 Existen varios lenguajes de programación (Pascal, C, haskell, Java, etc)

qué utilizamos python. La razón principal es que se encuentra instalado en todas las computadoras que el Plan Ceibal les entrega a los alumnos que cursan enseñanza media. Esto lo convierte en una herramienta al alcance de estudiantes y profesores ayudando a lograr los objetivos de la pasantía, es decir, integrar la algoritmia y la programación a los cursos de matemática. Puede también instalarse el intérprete python en cualquier otra computadora.

Además, existe en las últimas versiones de GeoGebra un intérprete python que interactúa con los objetos que se estén usando y permite ejecutar comandos y módulos sobre esos objetos. Esta facilidad es muy reciente aún y no se ha desarrollado del todo y por eso no la hemos incluido en este trabajo. Tampoco existe bibliografía suficiente sobre el tema, pero creemos que en un futuro próximo será de gran utilidad y al haber usado python en esta pasantía podremos aprovecharlo rápidamente. Pasamos ahora a describir cómo trabajar con el intérprete python.

Para comenzar abrimos el intérprete python en nuestra pantalla, vamos al menú principal y solo hacemos click un par de veces en: Aplicaciones > Programación > Python Ya tenemos una ventana lista para recibir órdenes en lenguaje python, solamente hay que escribir y pulsar Enter.




Ahora aprendamos algo del intérprete del lenguaje python. En la columna de la izquierda aparecen los comando que deben escribirse en la pantalla y luego pulsar enter, también aparecerán las respuesta que la computadora escribe en la pantalla, mientras que en la columna de la derecha aclaraciones y comentarios sobre los comandos. El intérprete funciona como un evaluador de expresiones:

en python

```
>>> 2 + 2
4
```

en español

¿cuánto es 2+2?

```
>>>4 == 3
False
```

¿4 es igual a 3?

Con el comando print podemos pedirle al intérprete que escriba lo que queramos:

| | |
|---|---|
| >>> print "hola mundo" | escriba: hola mundo |
| hola mundo | inmediatamente obedece<br>observemos que el texto va entre comillas |
| >>> print "el area es ", 2*5<br>el area es 10 | Puedes probar varios ejemplos para entender la sintaxis del comando *print* (por ejemplo la coma antes de 2 * 5)<br>NO UTILICES TILDES |
| >>> print "el volumen: ", 2**3, "cm3"<br>el volumen: 27 cm3 | |

Las variables son muy importantes en matemática y en programación, hay variables de muchos tipos, numéricas, alfanuméricas, listas, boolenas, caracteres, etc... comencemos a darle valor a una variable.

| | |
|---|---|
| >>> a = 5 | la variable a vale 5, esta línea es una asignación |
| >>> a<br>5 | para ver que esto es cierto... |
| >>> a == 4 | ¿es a igual a 4? |
| False | no lo es |

## La igualdad y la asignación

En matemática cuando queremos verificar que dos expresiones denotan el mismo valor, usamos el signo de =. El resultado de la verificación es un valor booleano: verdadero o falso. Por ejemplo, ¿cómo se procede para al evaluar 3 + 1 = 2 + 2? Se realizan las operaciones para obtener la forma canónica de las expresiones en ambos miembros de la igualdad, y si es la misma, el resultado es verdadero, si no es falso.

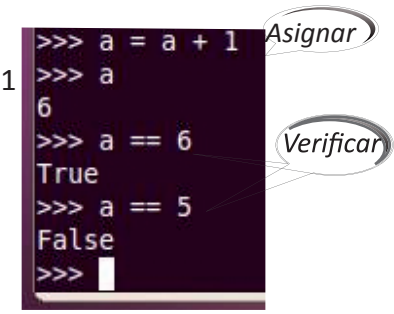
Pero el signo de = se usa en matemática también con otro significado: cuando decimos "para x = 3 hallar 2 x + 1", por ejemplo, pretendemos sustituir a la variable x por 3 en la expresión y hallar su valor. Esta operación de sustituir a x por 3, se denomina asignación y a pesar de que es muy distinta de la verificación de igualdad, en matemática se usa el mismo signo. Esto en programación es inaceptable, dado que la computadora debe saber qué hacer (o *verificar o asignar*) y por lo tanto los lenguajes de programación utilizan diferentes símbolos. En python, para la verificación de igualdad se usa el signo == mientras que para la asignación se usa el =.

```
>>> a = 5
```
Se asigna 5 a la variable a.

```
>>> a = a + 1
```
El valor de a (que es 5) se incrementa en 1

```
>>> a
6
```
Ahora es...

```
>>> a == 6
True
>>> a == 5
False
```




Es decir que en la asignación (=) se evalúa la expresión a la derecha del = (con el valor anterior de la variable a) y el resultado se asigna a la variable a (a la izquierda del signo de =).

El valor de una variable puede ser asignado por el usuario desde el teclado. Para ello se utiliza el comando input, veamos un ejemplo.
Deseamos que el usuario pueda indicar el valor de una variable a la que llamaremos n.

```
>>> n = input ("ingrese número mayor que 0: ")
```
Se despliega en la pantalla el mensaje *"ingrese número mayor que 0: "*
(Observar que NO ponemos tildes)

```
ingrese número mayor que 0: _
```
python espera a que el usuario escriba un número y pulse Enter
Una vez que el usuario ingresó un valor, python lo asigna a la variable n.

**RESUMIENDO:**

hay dos formas de asignar valores a las variables,
o bien directamente en el programa
o bien ingresados por el usuario.
Olvidar asignar valores a variables es
causa frecuente de errores.

## Un programa

Para hacer un programa, escribiremos primero todas las instrucciones (comandos) que debe realizar la máquina, en el orden adecuado y luego decimos al intérprete python que las ejecute.

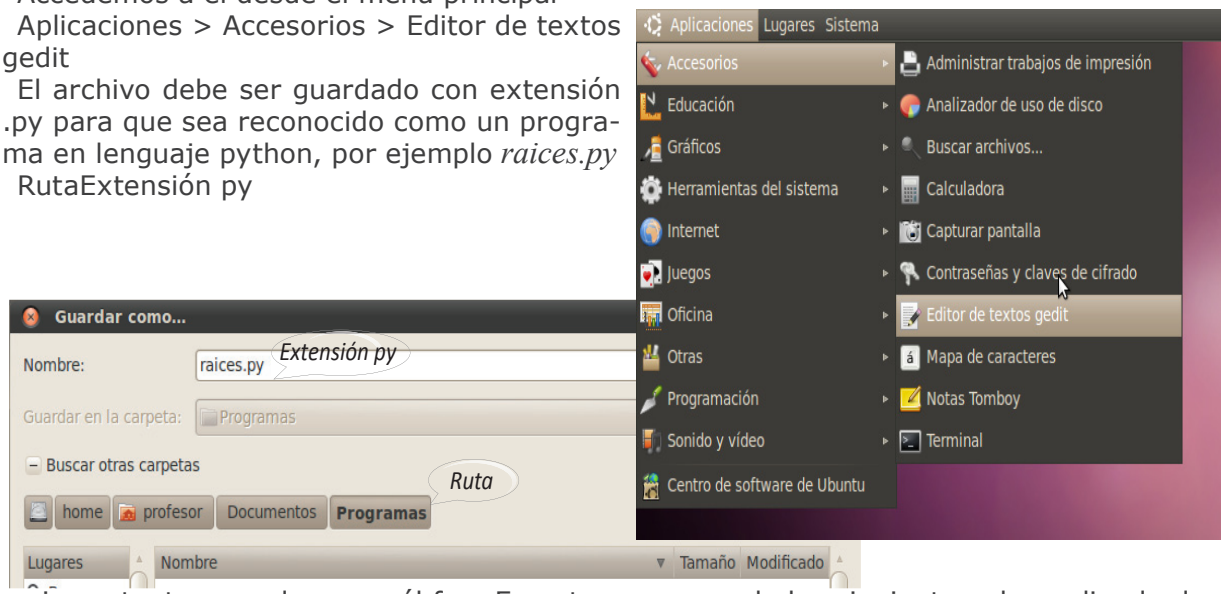
Entonces para empezar debemos utilizar un editor de texto. En esta ocasión es recomendable utilizar gedit, veremos que gedit es capaz de reconocer palabras que nosotros escribimos y destacar con diferentes colores comandos, funciones, números, los comentarios, etc...

Accedemos a él desde el menú principal

Aplicaciones > Accesorios > Editor de textos gedit

El archivo debe ser guardado con extensión .py para que sea reconocido como un programa en lenguaje python, por ejemplo *raices.py*

RutaExtensión py

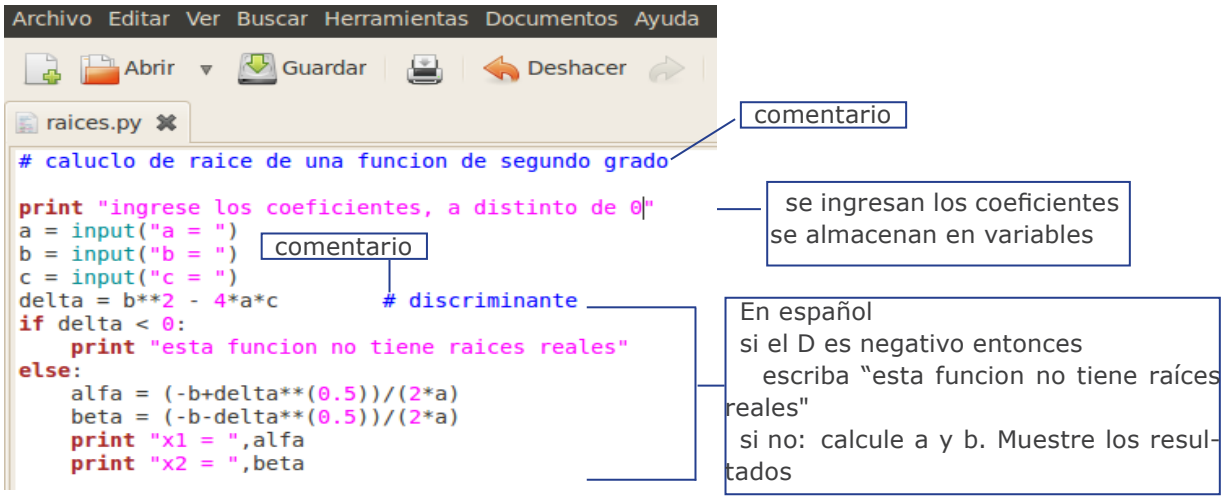



es importante recordar en cuál fue. En este caso, uno de los siguientes, dependiendo de si es profesor o estudiante:

*/home/Profesor/Documentos/Programas* *(/home/estudiante/matematica)*

Después de escribir el programa y guardarlo tendremos un documento como el que vemos a continuación. Los colores en las palabras se ven después de haber guardado el archivo con extensión .py

Explicaremos cada paso, pero debes hacerlo utilizando el editor para asegurarte de que estás cuidando cada detalle. Recordemos que son instrucciones para una computadora y cualquier diferencia provocará un mal entendido y la culparemos "injustamente" por no hacer lo que queremos.




* Comentar el programa

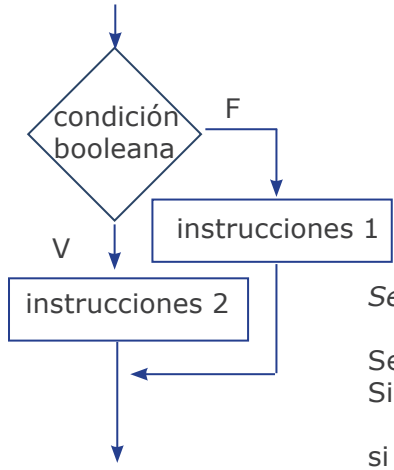
El signo # sirve para indicar que empieza un comentario, se puede escribir lo necesario para entender el programa. Puede usarse una línea entera o escribir a la derecha de una instrucción.

* El comando if then ... else ...

Para hallar las raíces debemos calcular la raíz cuadrada del discriminante, pero si éste es negativo, no podremos (porque buscamos raíces reales). Los lenguajes de programación proveen *instrucciones de selección* que permiten seleccionar qué instrucciones ejecutar dependiendo de una condición booleana. En python tenemos (observar que then se sustituye por :) :

*Sintaxis*

if *condición-booleana:*
*instrucciones 2*
else:
*instrucciones 1*



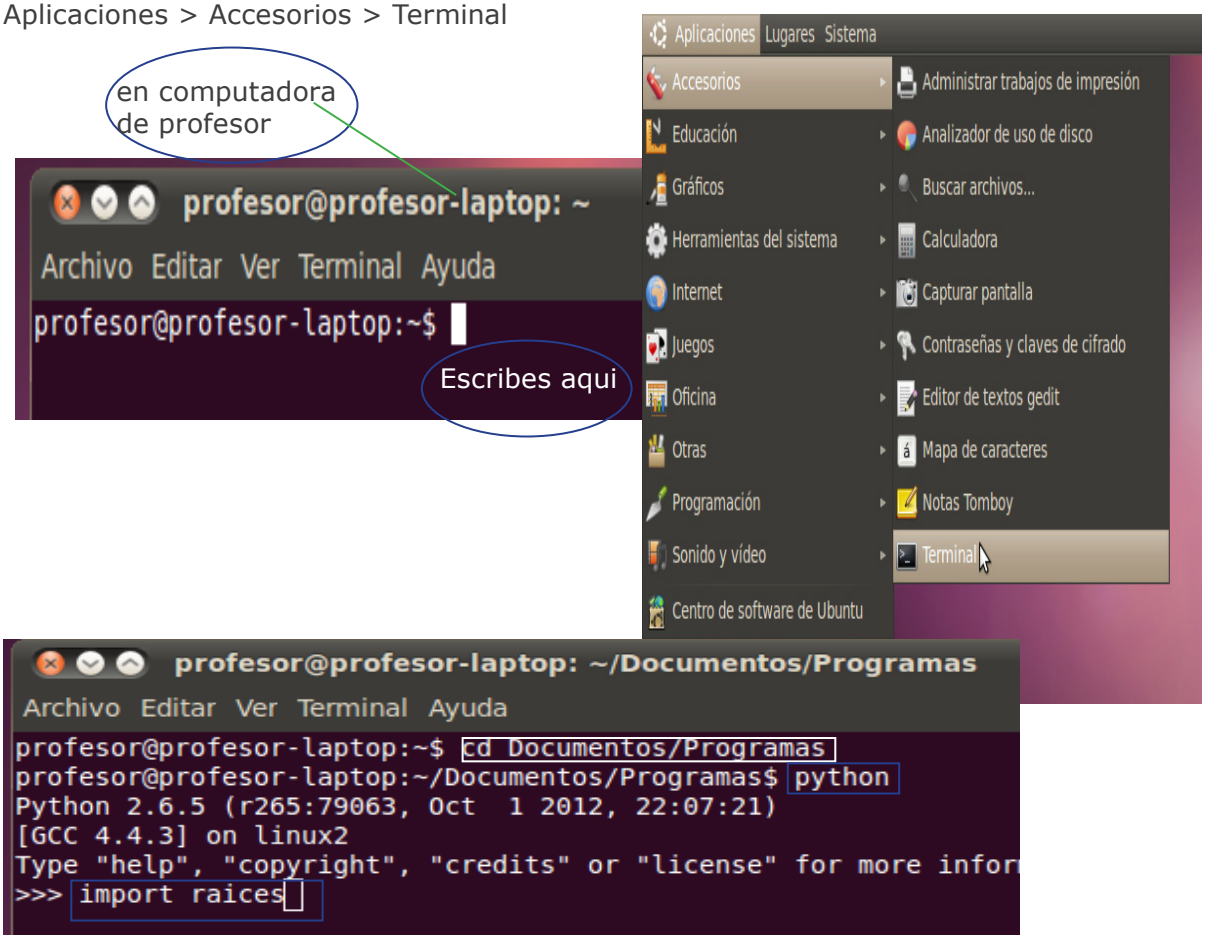

*Semántica*

Se evalúa la condición booleana
Si es Verdadera
se ejecutan las *instrucciones 2*
si es Falsa
se ejecutan las *instrucciones 1*

## Veamos ahora como funciona el programa

Ahora es el momento de ejecutar el programa que ya está escrito y listo para decirle a la computadora que cumpla con estas instrucciones y el intérprete python será de mucha ayuda.

Primero abriremos una ventana de terminal, esto es fácil de hacer desde el menú principal. Aplicaciones > Accesorios > Terminal







Una vez que tenemos el terminal vamos al directorio donde habíamos almacenado el archivo. Escribimos uno de los siguientes dependiendo de si es profesor o estudiante:

**cd Documentos/Programas** **(cd /home/estudiante/matematica)**
Llamamos al intérprete

**python**

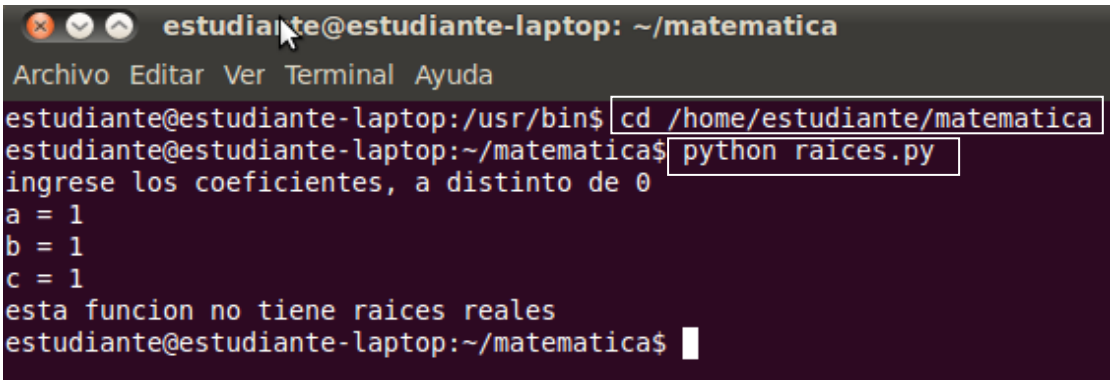
Ejecutamos nuestro programa
**import raices**
Ahora comienzan a ejecutarse las órdenes una por una ...
Si esto no ocurre hay algún error.
No debes desesperarte.
Nos ha pasado también.

```
>>> import raices
ingrese los coeficientes, a distinto de 0
a = 1
b = 1
c = 1
esta funcion no tiene raices reales
>>> reload (raices)
ingrese los coeficientes, a distinto de 0
a = 1
b = -5
c = 4
x1 =  4.0
x2 =  1.0
<module 'raices' from 'raices.pyc'>
>>> reload (raices)
ingrese los coeficientes, a distinto de 0
a = 1
b = 4
c = 4
x1 =  -2.0
x2 =  -2.0
<module 'raices' from 'raices.pyc'>
>>>
```

El programa pide que ingresemos los coeficientes y devuelve por escrito las raíces llamándolas x1 y x2. En caso de que la función no tenga raíces escribe en pantalla
"esta funcion no tiene raices reales"
Al finalizar queda el intérprete esperando...
>>>_
Para volver a ejecutar el programa lo recargamos...
>>> reload (raices)
Observemos que si hay una raíz doble, el programa escribe dos veces la misma raíz.
x1 = -2.0
x2 = -2.
*

*modificaciones para distinguir el caso de raíz doble*

```
if delta < 0:
    print "esta funcion no tiene raices reales"
else:
    if delta > 0:
        alfa = (-b+delta**(0.5))/(2*a)
        beta = (-b-delta**(0.5))/(2*a)
        print "x1 = ",alfa
        print "x2 = ",beta
    else:   # delta es 0
        raiz_doble = (-b+delta**(0.5))/(2*a)
        print "esta función tiene una raiz dolbe:"
        print "x = ", raiz _doble
```




* Otro Camino

También podemos ejecutarlo desde el terminal. Directamente, es decir, sin ingresar al intérprete en un sola orden escribiendo :

**python raices.py**

Al terminar la aplicación nos queda abierta la terminal pero no está en funcionamiento el intérprete python.

## Trabajamos con funciones

Un algoritmo puede contener instrucciones englobadas en forma de subalgoritmos para resolver partes del problema. De esta forma el algoritmo (o sea una solución al problema) adquiere una cierta estructura que permite visualizarlo mejor, detectar errores más rápidamente y modificarlo más fácilmente. Asimismo se vuelve más legible. Los subalgoritmos pueden implementarse en python como funciones.

* Sintaxis de la definición de función

```
def nombre_de_funcion (entrada[1]):
    cuerpo
    return salida
```

Si en el cuerpo se realizan cálculos sencillos, pueden devolverse directamente en salida (ver definición de delta abajo)

1 Entre paréntesis se escribe el valor de la/las variable/s, en programación se le llama parámetro o argumento.

```python
# coeficientes -> discriminante
def delta(a,b,c):
    return b**2 - 4*a*c

# Cuantas raices tiene?
# coeficientes -> cantidad de raices reales
def cant (a,b,c):
    d = delta (a,b,c)
    if d < 0:
        return 0
    else:
        if d > 0:
            return 2
        else:
            return 1
# Cuales son las raices reales,
# coeficientes -> raices reales
def raices (a,b,c):
    c = cant (a,b,c)
    if c  != 0:
        alfa = (-b + d**(0.5))/(2*a)
        beta = (-b - d**(0.5))/(2*a)
        return (alfa,beta)
```


***Ejemplo 1:***

*Función* delta, utilizada para calcular el discriminante de una ecuación de segundo grado.

Entrada: Terna de números reales, (coeficientes)

*Cuerpo*: Cálculo de b2 – 4ac

*Salida:* Número real, resultado del cálculo anterior (discriminante)

***Ejemplo 2:***

*Función* cant, determina la cantidad de raíces reales de una ecuación de segundo grado.

Entrada: Terna de números reales, (coeficientes)

*Cuerpo:* Calcula delta aplicando la función anterior[1] a los valores $a, b$ y $c$. Según el valor de d (delta(a,b,c)), cant devuelve 0, 1 ó 2.

*Salida:* Uno de los tres números 0, 1 ó 2.

***Ejemplo 3:***

*Entrada:* Terna de números reales, (coeficientes)

*Cuerpo:* Calcula cant usando la función anterior[2] aplicada a los coeficientes.

*Salida:* Par de números reales (las dos raíces reales)

Grabaremos el archivo de las funciones con el nombre "mod_raices.py" para luego utilizarlo como veremos en la próxima sección.

1 Observa que las funciones delta/cant se encuentran en este mismo archivo (módulo).

2Observa que las funciones delta/cant se encuentran en este mismo archivo (módulo).

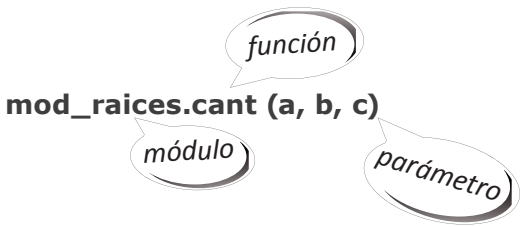
## Cómo utilizar las funciones desde otro módulo

Escribiremos en otro archivo el siguiente módulo en lenguaje python. Éste será un programa que utilizará las funciones del módulo anterior.

```
import mod_raices
print "ingrese los coeficientes (a distinto de 0)"
print "indique S para finalizar o N para seguir"
fin = 'N'
while (fin != 'S'):
    a = input ("a = ")
    if a == 0:
        print "no es ecuacion de 2do. grado"
        fin = raw_input ("finaliza? ")
    else:
        b = input ("b = ")
        c = input ("c = ")
        if mod_raices.cant (a,b,c) == 2:
            rr = mod_raices.raices(a,b,c)
            print "esta funcion tiene dos raices reales"
            print "x1 = ", rr [0]
            print "x2 = ", rr [1]
        else:
            if mod_raices.cant (a,b,c) == 1:
                print "esta funcion tiene una raiz doble"
                print "x = ", mod_raices.raices(a,b,c)[0]
            else:
                print "esta funcion no tiene raices"
        fin = raw_input ("finaliza? ")
print "Gracias por usar mi programa!"
```

**Import mod_raices**

Con este comando le decimos a la computadora que vamos a utilizar las funciones que están definidas en el módulo "mod_raices.py" como se ve en la linea



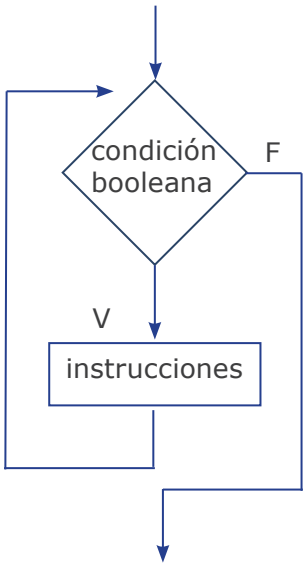

*rr* es un par, *rr [0] y rr [1]* son la primera y segunda componente del par respectivamente.

* El comando while

Se utiliza para repetir varias veces un mismo conjunto de instrucciones, como se puede ver en el ejemplo anterior.

*Sintaxis*

while *condición-booleana:*
*instrucciones*




*Semántica*

1. Se evalúa la condición booleana
Si es verdadera
se ejecutan las instrucciones
se vuelve al paso 1
Si no, finaliza.

## Funciones sobre divisibilidad

Este módulo contiene varias funciones utilizadas en divisibilidad. Podemos escribirlo y guardarlo con el nombre "divis.py" para luego poder utilizar cualquiera de estas funciones.

```
def divisores (a):
    return [x for x in range (1,a+1) if a%x == 0]

def esprimo (a):
    d = divisores (a)
    return d == [1,a]

def inter (l1,l2):
    l = []
    for element in l1:
        if element in l2:
            l.append (element)
    return l

def divisorescom (a,b):
    d1 = divisores (a)
    d2 = divisores (b)
    dc = inter (d1,d2)
    return dc

def mcd (a,b):
    dc = divisorescom (a,b)
    m = max (dc)
    return m

# esta es una funcion recursiva
# utiliza el algoritmo de euclides
def MCDr (a,b):
    r = a%b
    if r == 0:
        return b
    else:
    return MCDr (b,r)
```

Veamos algunos ejemplos en el intérprete python

¿7 es número primo?

También...

```
>>> import divis
>>> divis.esprimo(7)
True
>>>
```

¿Cuál es el MCD entre 6 y 15?

¿Cuáles son los divisores de 12?

¿Cuáles son los divisores comunes de 24 y 20?

```
>>> import divis
>>> divis.mcd(6,15)
3
>>> divis.divisores(12)
[1, 2, 3, 4, 6, 12]
>>> divis.divisorescom(24,20)
[1, 2, 4]
>>>
```

¿Cuáles son los números primos entre 1 y 30?

```
>>> [x for x in range (1, 31) if divis.esprimo(x)]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
>>>
```

## Buenas prácticas de programación

Pensar en integrar la programación en los cursos de matemática, nos lleva a decir unas palabras sobre la actividad de programar. Suele suceder que una vez que una persona aprende las nociones básicas de un lenguaje de programación, y comienza a ejecutar programas, se genera una especie de "círculo vicioso" por el cual la persona intenta corregir errores o mejorar su programa directamente en la máquina, y su trabajo creativo de diseño de la solución del problema, queda relegado. Esto tiene varias desventajas, destacaremos dos: por un lado, la corrección de un error suele generar otros en otras partes del programa (que puede consistir de varios módulos), y por otro lado, se produce un desfasaje entre la implementación y el diseño original, y suele ser difícil de reconstruir la solución definitiva. Lo que es deseable y constituye una buena práctica de programación recomendada por los académicos, es que frente a errores, se vuelva al diseño y se corrijan los errores en la solución elaborada. Luego, se corrija la implementación.

Otra recomendación consiste en conocer las herramientas que provee el lenguaje utilizado y usarlas correctamente. Muchas veces, un mismo resultado se logra utilizando distintas herramientas, y en algunos casos, se logra el resultado esperado pero utilizándolas mal.

Ejemplo: sea el problema: dado un número n, hallar los múltiplos de n menores o iguales que una cierta cota (en este caso la cota es 19*$n$). Un programa en python podría ser:

```
n = input("ingrese numero:")      # se ingresa n
i = 0                             # partimos de 0
r = []                            # resultado
while i <= 19:                    # iteramos hasta 19
    m = n*i                       # m es multiplos de n
    r  = r + [m]                  # se agrega al resultado
    i = i + 1                     # nueva iteracion
print r                           # cuando i = 20 mostramos el resultado
```

Funciona. Sin embargo python provee otras herramientas que permiten hacer un programa más cercano a la definición matemática de múltiplo, más corto, más sencillo y más eficiente:

```
n = input("ingrese numero:")    # se ingresa n
r = [n*x for x in range (20)]   # el resultado es la lista de n*x para 0 <= x < 20
print r
```

El exceso de comentarios es para ilustrar, conviene usar menos

Consideremos este segmento de programa:

```
def raices (a,b,c):
    delta = math.sqrt (b**2 - 4*a*c)
    if delta > 0:
        r1 = (-b + delta) / (2*a)
        r2 = (-b - delta) / (2*a)
    return (r1, r2)
```

El mismo tiene un error *en el algoritmo*, que es que denomina delta *a la raíz* de (b2 - 4 * a * c), por lo tanto si ese valor es negativo el programa devolverá un error. Es recomendable revisar el diseño y corregir el error en él, en vez de intentar corregir directamente en el programa. Muchas veces los errores son más fáciles de encontrar en el diseño del algoritmo que en el programa, donde muchas veces (la mayoría) una corrección puede generar otros errores y se vuelve difícil la corrección.

## Posibles errores

Es posible que cometas algunos errores en el momento de implementar en lenguaje python tus programas y funciones, por eso te contamos aquí cuáles fueron las cosas que hicimos mal y cómo el intérprete nos lo *dijo.*

```
SyntaxError: invalid syntax
ImportError: No module named raices
SyntaxError: invalid syntax
SyntaxError: EOL while scanning string literal
```

| | |
|---|---|
| No poner tildes.<br><br>Ejemplo:<br><br>print "raíces" | SyntaxError: Non-ASCII character<br><br>  <br> |
| Olvidé la indentación,<br>es decir no hice la tabulación<br>de forma adecuada | IndentationError: unexpected indent |

Ejemplo:

```
while i > 1:
m = m * i     las instrucciones están en el mismo nivel de
i = i +1      indentación que while
```

Olvidé los dos puntos
Ejemplo:

```
if a > 0
    m = m * n
```

SyntaxError: invalid syntax


Olvidé de los paréntesis en reload

SyntaxError: invalid syntax

Puse el archivo en otra carpeta

```
>>> import raices
ImportError: No module named raices
```

Olvidé cerrar las comillas literal

SyntaxError: EOL while scanning string

Ejemplo:

```
print "ingrese número
```


## Operaciones aritméticas

- Resta: -
- Suma: +
- Multiplicación: *
- División entera:

  cociente: //
  resto (módulo): %
  (2 // 3 es 0, 2 % 3 es 2)

- División (para reales): / (2.0 / 3.0 es 0.66666)
- Exponente: ** (a **b es $a^b$)

## Expresiones booleanas

- **Constantes:** True, False
- **Operadores:**

  and - conjunción
  or - disyunción
  not – negación

## Operadores relacionales

== (igual)
!= (distinto)
<= (≤, menor o igual)
>= (≥, mayor o igual)
> (mayor)
< (menor)

Para información sobre funciones predefinidas en python ver
http://mundogeek.net/tutorial-python/

# Correos de Contacto

Sylvia da Rosa
darosa@fing.edu.uy

Santiago Martorell
Santiago_6490@hotmail.com

Luis Langon
luislangon@adinet.com.uy

Daniela Pagés
danielapages@gmail.com

Teresa Pérez
tperezan@gmail.com

Teresita Carrión
pitacar@gmail.com

Patricia Añón
apifersa@hotmail.com

Santiago Vigo
svigo@adinet.com.uy

Franco Vuan Lockhart
fvuan@adinet.com.uy

Diseño - malepacomunicacion@gmail.com