## Referencia rápida 3 (MatUAM)
## Variables y expresiones booleanas. Control del flujo

**Variables y expresiones booleanas.**

1. Un tipo especial de variable es el *booleano* o *lógico*, que solo toma dos valores **True** o **true** para *verdadero*, y **False** o **false** para *falso*. La función **bool**() devuelve una valor booleano al evaluar su contenido. Si este, en su tipo o ambiente, es en algún sentido *neutro*, **bool**() devolverá **False**, y **True** en caso contrario:

```
bool(0)                          False
bool(8)                          True
bool([])                         False
bool(dict())                     False
bool(set('abc'))                 True
L=[1,2]; M=L.reverse();bool(M)   False
```

2. Las operaciones básicas con variables booleanas son la *conjunción* (**and**), las *disyunción* (**or**) y la *negación* (**not**):

| and | True | False |
|---|---|---|
| True | True | False |
| False | False | False |

| or | True | False |
|---|---|---|
| True | True | True |
| False | True | False |

| | not |
|---|---|
| True | False |
| False | True |

3. **Expresiones booleanas**: Decimos que una *expresión* o *función* es *booleana* si al ejecutarse devuelve uno de los dos valores booleanos. Ejemplos típicos de expresiones booleanas son
   a) elemento **in** contenedor, que corresponde a la relación de *pertenencia*. La no pertenencia se indica con **not in**.
   b) La comparaciones:
      - A==B, es decir, A y B son *idénticos* al evaluarse, y se puede aplicar a números, estructuras de datos, o cualquier otro tipo de objetos. La no identidad se indica con != o con <>.
      - A<B, A<=B, A>B, A>=B, que se aplica a objetos para los que hay un orden natural.
   c) Muchas funciones predefinidas en Sagemath, como **all**() o **any**() devuelven un valor booleano. Y en particular las que tienen prefijo .**is_** o .**is**, como .**is_prime**(), .**is_irreducible**(), .**issubset**(), ...
   d) Operaciones con variables o expresiones boolenas.

```
7 in prime_range(100)                          True
(1==1.) and (2>1)                              True
'pera'< 'manzana'                              False
2>=4/2                                         True
1!='1'                                         True
'35'in '345'                                   False
'0'not in '9870'                               False
all([(2^j-1).is_prime for j in [2,3,5,7,11]])  True
```

**Control de flujo.** Muchas estructuras de datos son *iterables*, es decir, podemos crear un bucle que recorra uno por uno los elementos de la estructura de datos y para cada uno de esos elementos ejecute un bloque de instrucciones. Todas las estructuras de datos que ya conocemos, listas, tuplas, cadenas de caracteres, conjuntos y diccionarios, son iterables.

4. **Bucles for**: se repite la ejecución de un bloque de código un NÚMERO DE VECES DETERMINADO. La sintaxis *básica* de un bucle **for** es

```
for <elemento> in <contenedor>:
    instrucciones ...
```

donde:

- El caracter ":" marca el final del encabezamiento del bloque.
- El bloque de instrucciones interno a repetir aparece *indentado* o *sangrado* con un número fijo de espacios en blanco. El editor, al cambiar de línea tras los dos puntos, ubica el cursor con el sangrado adecuado.
- El bloque interno repetirá su lista completa *interna* de instrucciones **len**(contenedor) veces. Además, en cada iteración, <elemento> tendrá asignado, uno a uno, los elementos del contenedor. Este valor puede utilizarse, si conviene, en las instrucciones internas.
- El final del bloque lo marcará la primera línea que aparezca no indentada.
- Al salir del bloque, la asignación se mantendrá con el último valor.

```
m=96
Fm=str(factorial(m))
for digito in '0123456789':
    print '%s aparece %d veces en %d!'%(digito,Fm.count(digito),m)
```

```
0 aparece 36 veces en 96!
1 aparece 9 veces en 96!
2 aparece 7 veces en 96!
3 aparece 9 veces en 96!
4 aparece 16 veces en 96!
5 aparece 11 veces en 96!
6 aparece 11 veces en 96!
7 aparece 15 veces en 96!
8 aparece 16 veces en 96!
9 aparece 20 veces en 96!
```

5. **Los bloques if** <condicion>: Un **if** sirve para *ramificar la ejecución de un programa*: cada uno de las partes del **if** define un camino alternativo y el programa entra o no según se verifique o no una condición. La sintaxis del **if** es:

```
if <condición 1>:
    instrucciones...
elif <condición 2>:
    instrucciones...
else:
    instrucciones...
```

Las diversas condiciones deben ser *booleanas* y puede haber tantas líneas **elif** como queramos.

La última parte, el **else** es opcional y lo usamos si necesitamos indicar qué debe hacer el programa en el caso en que no se cumpla ninguna de las condiciones.

```
numero=randint(1,1000)
if numero%12 in [0,2,4]:
    numero+=1
elif numero%12 in [1,3,5,7]:
    numero=2*numero-1
elif numero%12 in [6,8]:
    numero=numero/2
else:
    numero*=2
print numero
```

```
581
```

6. **Bucles while**: son similares a los **for**, pero los usamos cuando no sabemos, a priori, cuantas iteraciones debe dar el bucle, aunque *estamos seguros de que acabará*. Su sintaxis es:

```
while <condición>:
    instrucciones...
```

En este tipo de bucles, solo se entra a la parte interna si la condición, una expresión booleana, es **True**, y el bucle se repetirá siempre que esta condición siga siendo verdadera. Para no producir un bucle infinito, es **imperativo** que entre las instrucciones del interior en alguna de las iteraciones esta condición cambie a **False**, momento en el cual el bucle deja de ejecutarse y la ejecución seguirá por el código posterior fuera del bucle.

Siempre hay que tener mucho cuidado con los bucles **while** infinitos ya que, aparte de no producir ningún resultado, frecuentemente cuelgan la máquina. Un ejemplo evidente de bucle infinito sería algo como **while** 5>4:....

```
m,tope=1,25
divisores=len(m.divisors())
while divisores<tope:
    m+=1
    divisores=len(m.divisors())
print 'El primer número que tiene al menos %d divisores es: %d.'%(tope,m)
```

```
El primer número que tiene al menos 25 divisores es: 720.
```

7. Por último, los bucles se pueden anidar, es decir, cualquier bucle puede entrar a formar parte del conjunto de instrucciones de otro, sangrando adecuadamente las líneas de los bucles interiores.