INTRODUCCION A PYTHON

PYTHON NO REQUIERE DECLARAR TIPO DE LAS VARIABLES

EN JAVA:

```
int n=12;
n=n+1.5;

//error: "posible perdida
de precisión".
```

EN PYTHON:

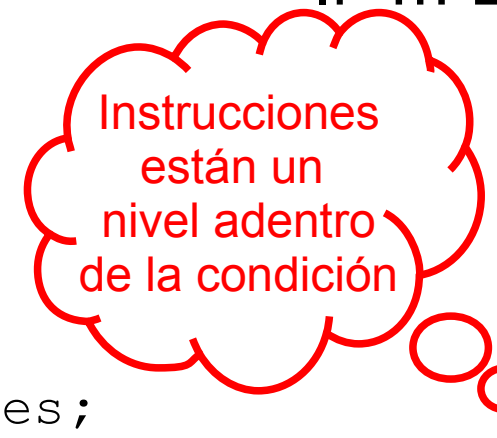
```
n=12
n=n+1.5

#correcto: el nuevo valor de
n es 13.5
```

Las sentencias no terminan en ;

se usa para hacer comentarios

IF … ELSE



EN JAVA:

```
if (condición){
    instrucciones;
}
else if(condición){
    instrucciones;
}
else{
    instrucciones;
}
```

```
//java usa llaves para
reconocer inicio y fin de
instrucciones.
```

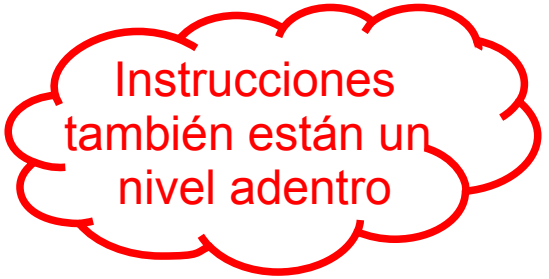
EN PYTHON:

```
if condición:
    Instrucciones
elif condición:
    Instrucciones
else:
    Instrucciones
```

```
#python usa ":" para indicar
inicio de instrucciones e
indentación para reconocer
hasta donde llegan.
```

USO DE CICLOS - WHILE

EN JAVA:




```
while (condición){
    instrucciones;
}

System.out.print("salimos del ciclo");

//java usa llaves para reconocer inicio y fin de instrucciones.
```

EN PYTHON:

```
while condicion:
    Instrucción 1
    Instrucción 2
print "salimos del ciclo"

#python usa ":" para indicar inicio de instrucciones e indentación para reconocer hasta donde llegan.
```

USO DE CICLOS - FOR

EN JAVA:

Instrucciones también están un nivel adentro

```
for (i=0;i<max;++i){
    instrucciones;
}
System.out.print("salimos del ciclo");
//java usa llaves para reconocer inicio y fin de instrucciones.
```

EN PYTHON:

```
for i in range(max):
    Instrucción 1
    Instrucción 2
print "salimos del ciclo"
#python usa ":" para indicar inicio de instrucciones e indentación para reconocer hasta donde llegan.
```

EJERCICIO

Escribir un programa que cuente hasta el número ingresado por el usuario. Si el número es 0 el programa debe terminar.

```
>>> ======= RESTART ========
>>>
ingrese numero: 5
1
2
3
4
5
ingrese numero: 2
1
2
ingrese numero: 3
1
2
3
ingrese numero: 0
fin del programa
>>>
```

SOLUCION

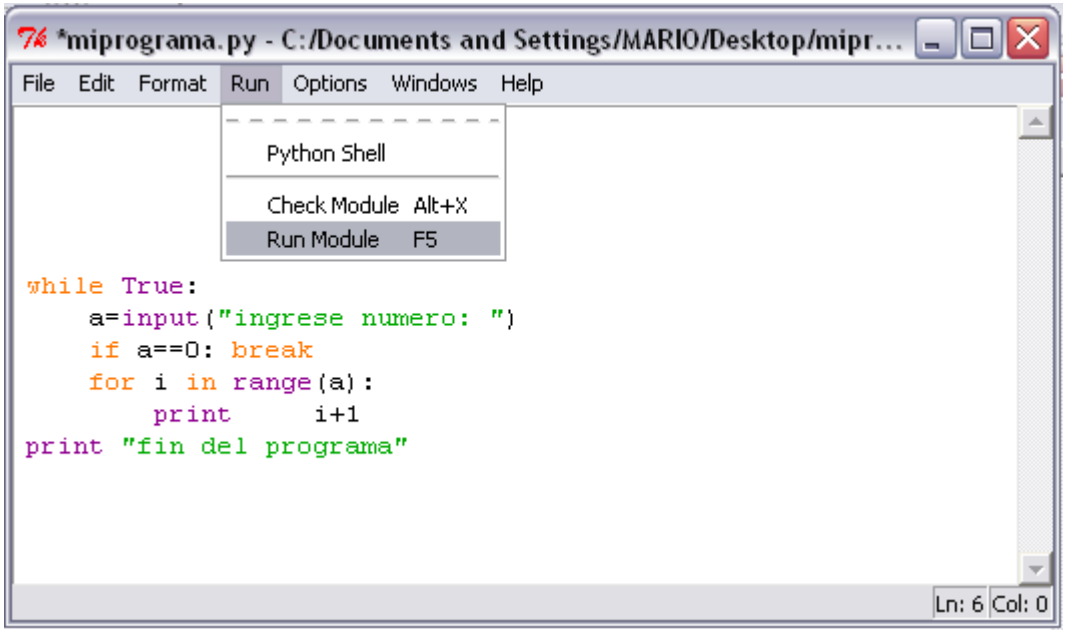
Valores Booleanos:
True False

Imprime el texto:
"ingrese un numero:"
Guarda en "a" el valor numérico ingresado por el usuario.

Condición de término

```
while True:
    a=input("ingrese numero: ")
    if a==0: break
    for i in range(a):
        print i+1
print "fin del programa"
```

SOLUCION


*miprograma.py - C:/Documents and Settings/MARIO/Desktop/mipr...
File Edit Format Run Options Windows Help
Python Shell
Check Module Alt+X
Run Module F5
while True:
a=input("ingrese numero: ")
if a==0: break
for i in range(a):
print i+1
print "fin del programa"
Ln: 6 Col: 0

FUNCIONES

Contenido de la función también está un nivel adentro del encabezado

La función puede no retornar nada.

```
def nombrefuncion(argumentos):
    Instrucciones
    return resultado
print "esto está fuera de la función"
```

EJERCICIO

Escriba una función que reciba dos números y retorne el promedio.

Use la función en un programa que solicite al usuario sus notas y muestre el promedio.

```
>>> ==== RESTART ====
>>>
ingrese nota 1: 6.5
ingrese nota 2: 5.3
promedio =  5.9
>>>
```

SOLUCION

El encabezado no requiere indicar tipo de retorno ni tipo de argumento

Esto está dentro de la función

Este es el programa (fuera de la función)

Llamada a la función

```
def promedio(numero1 ,numero2):
    return (numero1+numero2)/2.0

a=input("ingrese nota 1: ")
b=input("ingrese nota 2: ")
print "promedio = ", promedio(a,b)
```

CLASES Y OBJETOS

Encabezado de la clase

Constructor

Métodos

```
class Nombreclase:
    def __init__(self, argumentos):
        self.variabledeinstancia = valor
    def metodo1(self, argumentos):
        instrucciones
    def metodo2(self, argumentos):
        instrucciones
```

Las funciones pueden no tener argumentos pero siempre llevan "self"

CLASES Y OBJETOS: EJEMPLO

```
class Fraccion:
    def __init__(self, x=1, y=1):
        self.num = x
        self.den = y
    def valor(self):
        return 1.0*self.num/self.den
    def suma(self, f):
        numerador = self.num*f.den + self.den*f.num
        denominador = self.den*f.den
        return Fraccion(numerador, denominador)
```

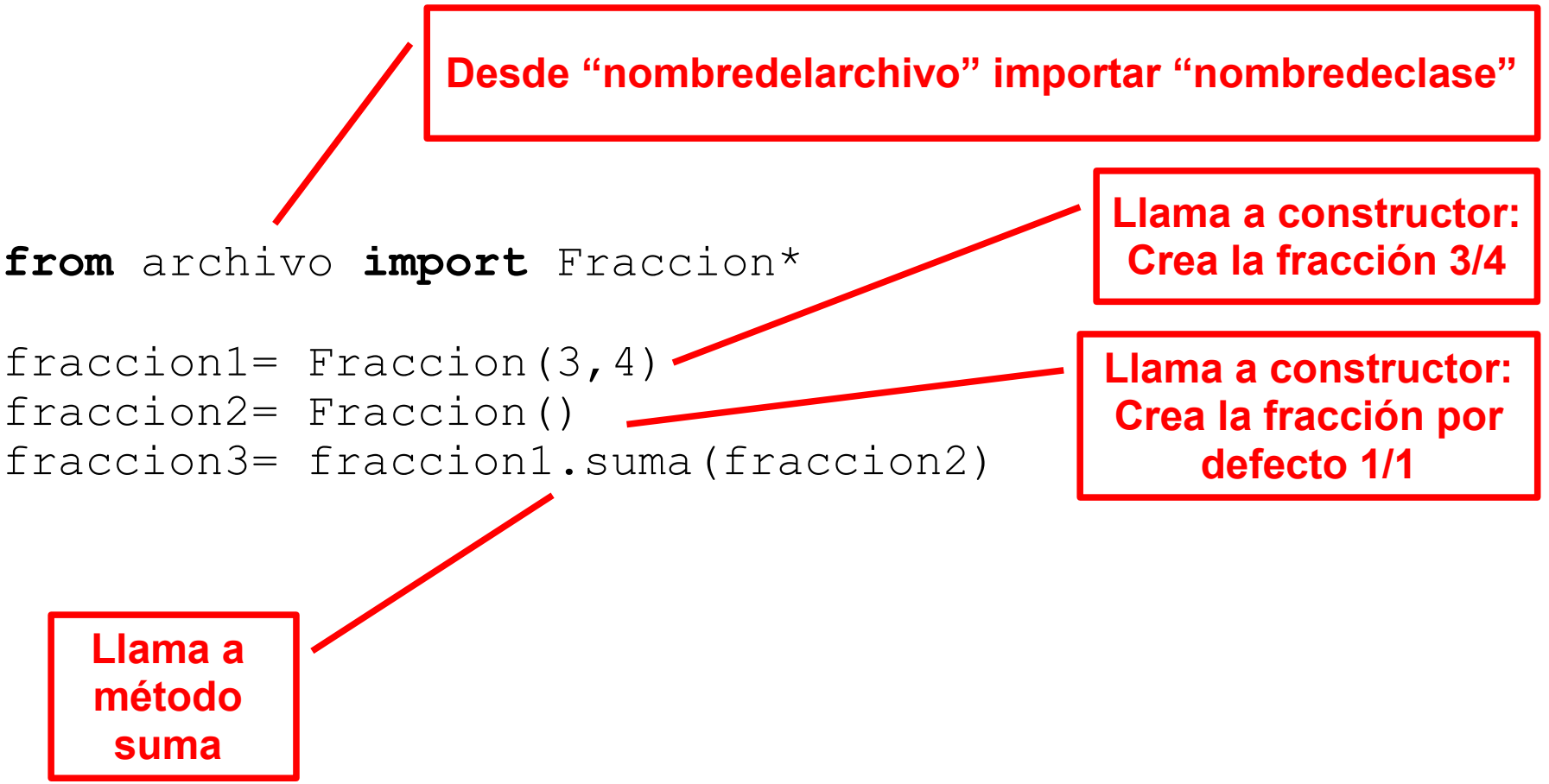
Valores por defecto

Método que no recibe argumentos

Método que recibe un objeto de la clase Fraccion como argumento.

Llama al constructor

CLASES Y OBJETOS: EJEMPLO DE USO



ENCABEZADOS PREDEFINIDOS QUE PERMITEN USAR OPERADORES COMO +, - , str(objeto), < , etc. ENTRE OBJETOS

```
#Sean f1 y f2 objetos de una clase.

def __add__(self,f):
#uso: en vez de: f1.__sum__(f2)
      Se usa directamente: f1 + f2

def __str__(self):
#uso: en vez de f1.__str__()
      Se usa directamente: str(f1)
      Nota:entrega string del objeto (ejemplo: "3/4")

def __gt__(self,f):
#uso: en vez de f1.__gt__(f2)
      Se usa directamente: f1>f2

# ... otros predefinidos ...
```

HERENCIA

```
class Madre:
    def __init__(self, arg):
        self.a = arg

class Hijo1(Madre):
    def metodo1(self): instrucciones
    def metodo2(self): instrucciones

class Hijo2(Madre):
    def __init__(self, arg1, arg 2)
        Madre.__init__(self,arg1)
        self.b = arg2
    def metodo1(self): instrucciones
    def metodo2(self): instrucciones
```

Constructor clase Madre

Indica que hereda de la clase Madre

Hereda el constructor de Madre (no se explicita)

Llama a constructor Madre y lo complementa con nuevos argumentos

OTROS ELEMENTOS ÚTILES

```
LISTAS:
a=[] #inicializacion lista vacía
a=[1,10,15] #inicializacion con elementos
a.append(20)# agrega 20 al final de la lista
a[i] # valor en indice i
len(a) #entrega cantidad de elementos en lista a

for elemento in a:
   #recorre cada elemento de la lista a
   # "elemento" toma los valores 1,10,15 y 20 en cada ciclo

Se pueden utilizar listas de objetos.


LISTAS DE LISTAS:
a=[[],[]] #inicializacion de lista de 2 listas vacías
a=[[1,10,15], [3,4,5]] #inicializacion con elementos
a[i][j] # valor en fila i, columna j
```