El nucleo de la programación

Temario de la clase

- Variables lógicas. True False.
- Bifurcaciones. If else.
- Variables bandera (flag).
- Loops.
- While.
- For.
- Índices contadores. Acumuladores.

Variables lógicas

Una variable lógica puede tomar dos valores: True o False.

```
>>>  lpreg=True
>>>  type(lpreg)
<type 'bool'>
```

Son de utilidad para switches, configuraciones de si se quiere que el código tenga determinadas características o no de acuerdo al interesado.
Se usan de máscara cuando se trabaja con datos.
Las tres operaciones de variables lógicas mas reconocidas: and, or y not.

```
>>>  lresp=False
>>>  lresp and lpreg
False
```

Operaciones combinadas (OJO con el orden!)

```
>>>  lcom=lresp and (lpreg or not lbe)
>>>  lcom
False
```

Las variables lógicas se asocian a 0 (False) y 1 (True) en python.

Operaciones con variables lógicas

Las tres operaciones de variables lógicas mas reconocidas: and, or y not.

```
>>> lresp=False
```

Operador and

```
>>> lresp and lpreg
False
```

True and True → True

True and False → False

False and False → False

Operador or

```
>>> lresp or lpreg
True
```

True or True → True

True or False → True

False or False → False

Operaciones con variables lógicas

Operador not

```
>>> not lpreg
False
```

Not True $\rightarrow$ False

Not False $\rightarrow$ True

Operaciones combinadas (OJO con el orden!)

```
>>> lcom=lresp and (lpreg or not lbe)
>>> lcom False
```

Operadores que resultan en variables lógicas

Es la variable `a` igual a la variable `b`? Simbolo utilizado para representarlo:

==

```
>>> lresp = a == b
```

Es la variable a distinta a la variable b? !=

```
>>> lresp = 1!=2
```

Es la variable a mayor a 5? >

```
>>> lresp = a > 5
```

```
>>> lresp = a >= 6
```

Combinación de operaciones:

```
>>> lresp = a >= 6 and a <=10
```

El resultado de todas estas operaciones es una variable lógica. True False

Operadores lógicos para cadena de carácteres

```
>>> s1 = 'bc'
>>> s2 = 'abcde'
```

El operador in pregunta si una cadena se encuentra en la otra:

```
>>> s1 in s2
```

El operador in not pregunta si una cadena no se encuentra en la otra:

```
>>> s1 in not s2
```

El operador is pregunta si una variable es la otra. `>>> x0 is y`

```
>>> x0 is None
```

Las variables las puedo definir como 'None'

Nota: `is` da True si las dos variables se refieren al mismo objeto en memoria.

`==` da True si las dos variables son iguales

Ejemplo:

```
In [9]: x=[1,2]
In [10]: y=[1,2]
In [11]: x is y
Out[11]: False
In [12]: x == y
Out[12]: True
```

La instrucción if: condicional

Hay muchas veces en un programa que vamos a querer controlar el flujo, es decir que el programa haga algo si la respuesta es afirmativa y que no lo haga si la respuesta es negativa:

```
>>> syes=raw_input("Desea terminar (s): ")
>>> if syes == 's':
...     print 'Respuesta s=si. Termino el programa'
...     raise SystemExit
```

La estructura de la instrucción if es:

if (variable lógica): Si la variable lógica es verdadera entonces:

(4 espacios en blanco) Hace esto.

Los espacios en blanco, tabulación, son parte de la instrucción. (No end)

La instrucción if-else

Si pasa esto, haga algo si no pasa eso haga otra cosa:

```
syes=input("Desea continuar (s/n): ")
if syes == 's':
    print 'Respuesta s=si continua.'
else:
    print 'Cualquier otra respuesta termina. '
    quit(). '
```

La estructura de la instrucción if-else es:

if (variable lógica): Si la variable lógica es verdadera entonces:

(4 espacios en blanco) Haga esto

else: Si la variable lógica es falsa entonces

(4 espacios en blanco) Haga esto otro

La instrucción if-else. Ejemplos.

Si queremos calcular raíces cuadradas a partir de un número que introduce el usuario, nos deberíamos asegurar que los números son positivos para que no haya error.

```
a=input('Introduzca el nro: ')
if a > 0:
    sqa=math.sqrt(a)
    print 'La raiz cuadrada del nro es:',sqa
else:
    print 'El nro debe ser positivo'
```

La instrucción if-else. Ejemplos.

El resultado lo podemos guardar en una variable lógica y luego usar la variable en el if.

```
a=float(input('Introduzca el nro: '))
lapos=a > 0
if lapos:
    sqa=math.sqrt(a)
    print ('La raiz cuadrada del nro es:',sqa)
else:
    print ('El nro debe ser positivo')
```

Condicionales anidados: elif.

Hay veces que necesitamos varias opciones no solo dos.

Para esto existe el elif.

Es una mezcla de else y de if, rp“de lo contrario si es que pasa esto”

```
a=input('Introduzca un nro: ')
if a == 0:
    print ('El nro es zero')
elif a> 0:
    print ('El nro es positivo')
else:
    print ('El nro es negativo')
```

Varias opciones elif. Ejemplo.

El elif es útil para cuando se le da opciones al usuario [No existe el case].

```
a=input('Introduzca un nro: ')
print 'Que desea calcular: '
opt=float(input('(1) Cuadrado, (2) Raiz cuadrada, (3) Logaritmo:'))
if opt == 1:
    print ('El cuadrado es: ',a**2)
elif opt == 2:
    print ('La raiz es: ',math.sqrt(a))
elif opt == 3:
    print ('El logaritmo es: ',math.log(a))
else:
    print 'Hay solo tres opciones 1,2,3'
```

En estos casos siempre conviene usar un else a lo último para cualquier problema que hubo en el ingreso de los datos (o cuando se esta ejecutando el programa),
Entonces estamos avisando de que “No se encontró ninguna opción válida”.

Ejemplo. Encontrar las raíces de una ecuación cuadrática

Guía 1 (1c). Describa un procedimiento mediante expresiones matemáticas y pasos a seguir para obtener los ceros de una función cuadrática.

```
print ('Determina raices reales de a x^2 + b x + c = 0')
a=float(input('Introduzca a: '))
Idem b y c

rad = b**2 - 4 * a * c
if rad < 0:
    print ('La ecuacion no tiene raices reales')
elif rad == 0:
    print ('La ecuacion tiene una raiz',-b/(2*a))
else:
    print ('Tiene dos raices: ')
    sqr= rad**0.5 /(2*a)
    raex=-b/(2 * a)
    print ('Radic. Positivo: ',raex + sqr)
    print ('Radic. Negativo: ',raex - sqr)
```

Variables bandera.

En numerosas situaciones queremos guardar el estado de una situación. Generalmente la variable bandera o flag tiene dos opciones 0 o 1. Puede usarse variable lógica.
En algun momento guardamos el estado de situación:

```
if nro % 2 == 0:
   band=1
else:
   band=0
```

En otro lugar del programa usamos el estado de situación de la variable:

```
if band == 1:
   print ('El numero es par')
```

Bucles/loops/ciclos.

Loop: Conjunto de instrucciones que necesitamos repetir una cantidad de veces.

- Si queremos contar las esferas rojas de la caja. Necesitamos recorrer todas las esferas.
- Si queremos evaluar a una función en un intervalo discreto. Tenemos que evaluarla en cada punto x_i.
- Si queremos saber la edad de los compañeros del curso. Necesitamos repetir la pregunta a todos los compañeros.
- Si una empresa tiene muchos clientes con deudas y quiere saber cual es el monto total de la deuda, necesita recorrer a todos los clientes y sumar cada una de las deudas.

Les llamamos bucles, ciclos, loops, etc.

Bucles con while.

El comando while hace que la computadora repita una serie de órdenes hasta que se cumpla una condición lógica.
Para producir un bucle de 8 iteraciones comenzando con i=1 hasta i=8:

```
i=0
sum=0
while sum<=80:
    print (i)
    i += 1
    sum += i
```

El while es solo para cuando no conocemos el número de ciclos.

Una forma simplificada (pythonica) de poner contadores en python:

```
i+=1
```

esto es exactamente lo mismo que

```
i=i+1
```

Notar que siempre después del while ... : siguen las tabulaciones hasta donde termina la serie de instrucciones que queremos se repitan.

Bucles con for

La instrucción mas importante para hacer bucles o repeticiones de órdenes es con for. Este se usa para tomar valores de una lista.

for i in lista de valores:

```
>>> a=['a','b','c']
>>> for char in a:
        print char,')'
a )
b )
c )
```

Otra forma muy utilizada es usando la generación de listas con range:

```
>>> for i in range(3):
...     print (i,')'}
0 )
1 )
2 )
```

Elementos e indice:

```
a=['a','b','c']
for i,car in enumerate(a):
        print (i,char,')')
```

Dos listas:

```
for nombre,direccion in
            zip(nombres,direcciones):
```

Bucles con for

Recordar que el range permite empezar de cualquier número y terminar, ej. range(2,10,2)
Si tenemos un range(2,5) comenzará en 2 pero terminará en 4! uno antes del número máximo, pero respetando que el número de ciclos es max-min (5-2=3).
Si queremos terminar el bucle, `for`, si se cumple alguna condición usamos `break`.

```
for i  in range(100):
     < calculos >
     if error:
          print 'Ocurrio un error en el bucle'
          break
     < mas calculos >
```

Si queremos cortar el ciclo usamos `continue`

¿Cuando uso for y cuando while?

Las instrucciones `for` y `while` hacen lo mismo aunque el `while` requiere una línea mas.

- Si el número de ciclos es fijo y conocido uso el `for`.
- Cuando el número de ciclos depende de una cantidad que tengo que calcular uso el `while`.

Si estas en duda es porque tenes que usar el `for`.

Prestamos hasta una cantidad máxima

Ejemplo 1. Supónganse que una empresa permite a sus clientes hasta 200.000 pesos de deudas y los clientes pueden ir comprando y endeudarse hasta ese máximo.

```
MontoMax=200000
icompra=0 # numero de compra/factura
MontoAdeudado=0
while  MontoAdeudado < MontoMax:
     icompra=icompra + 1  # cuento la cantidad de compras
     MontoFactura=float(input('Monto total de la factura adeudada'))
     MontoAdeudado = MontoAdeudado + MontoFactura

print ('El cliente hizo: ',icompra,' compras y debe: ',MontoAdeudado)
```

Este es de una empresa de buena fe. Si el empresario fuera desconfiado o mas estricto como debería adaptar el algoritmo?

¿En que tiempo se desarrolla la turbulencia en un fluido?

Ejemplo 2. Tenemos que resolver la dinámica de un fluido y decir para que tiempo se vuelve turbulento. El número de Richardson $Ri < 1/4$ indica que el fluido esta turbulento.
En este caso no sabemos cuantos ciclos vamos a tener que hacer. Vamos a requerir tantos ciclos como los necesarios para que el $Ri < 1/4$.

```
Ri=1.0
i=0
while  Ri >= 0.25:
     i=i + 1 # cuento la cantidad de tiempos
     u,v,T,rho,p=< funcion que calcula la dinamica >
     Ri = < funcion que calcula el Richardson > (depende de u y v)

print ('El fluido desarrollo turbulencia en: ',i * dt,' s')
```

Ejemplo 3: Evaluación de una función. Intervalo abierto

Evaluar la función $f(x) = x^2 + 4x - 2$ en un determinado intervalo $[a, b)$ con una resolución de Δx.

```
Ingresos de a,b y deltax
x=a
print ('El valor de la funcion x^2 + 4 x -2 es')
print ('   x,      y  ')
while  x < b:
    y = x**2 + 4 * x -2
    print (x,y) # poner con una precision de 3 digitos o lo que se requiera
    x = x + deltax
```

Ejemplo 4: Evaluación de una función. Intervalo cerrado

Evaluar la función $f(x) = x^2 + 4x - 2$ en un determinado intervalo $[a, b]$ usando n evaluaciones.

```
Ingresos de a,b y n
x=a
deltax=(b-a)/(n-1.)
print ('El valor de la funcion x^2 + 4 x -2 es')
print ('   x,      y  ')
for i in range(n):
    y = x**2 + 4 * x -2
    print (x,y)
    x = x + deltax
```

Ejemplo 5: Evaluación de una función. Intervalo cerrado

Evaluar la función $f(x) = x^2 + 4x - 2$ en un determinado intervalo $[a, b]$ con una resolución mínima de Δx.

Cuantos ciclos tengo que hacer? $n = \frac{b-a}{\Delta x} + 1$

Cual es el problema con esto?

```
Ingresos de a,b y deltax
x=a
n = int( (b - a)/deltax ) + 2
# me aseguro que haya mas resolucion de la requerida
deltax = (b-a)/(n-1.)        # calculo el salto exacto
print ('El valor de la funcion x^2 + 4 x -2 es')
print ('   x,      y  ')
for i in range(n):
    y = x**2 + 4 * x -2
    print (x,y)
    x = x + deltax
```

Uso del for con listas

Si tuvieramos una lista de clientes y queremos recorrer cada cliente:

```
clientes=['Laura', 'Eugenia','Elvira','Graciela']
for cliente in clientes:
    print('Nombre del cliente: ',cliente)
```

En el caso de dictionarios:

```
clientes=['Nombre':['Laura', 'Eugenia'],'Edad'=[25,38]]
for kcliente in clientes:
    print(kcliente) # key del dictionary
    print(clientes[kcliente]) # values de la key
```

Si quiero los valores directamente:

```
clientes=['Nombre':['Laura', 'Eugenia'],'Edad'=[25,38]]
for vcliente in clientes.values:
    print(vcliente) # values del dictionary
```

Contadores

Variable Entera que cuenta cuantas veces ocurre una situación.

Ejemplo: Cantidad de múltiplos de 2, entre 1 y un número n.

```
n=input('Ingrese el numero ')

j=0 # Inicializo el contador
for i in range(n):
    if i % 2 == 0:
        j = j + 1 # Cada vez que ocurre la condicion le agrego uno
print ('Hasta el numero ',n,'hay ',j,'multiplos de 2')
```

En python hay una forma corta para expresar el update de contadores

j = j + 1 $\rightarrow$ j + = 1

Significan exactamente lo mismo [Para mi es irrelevante cual usen].

Acumulador

Variables que acumulan resultados en forma repetitiva.

Ejemplo: Sumatoria de todos los números enteros menores o iguales a un número n, i.e. $S = \sum_{i=1}^{n} i$.

```
n=input('Ingrese el numero ')

sum=0 # Inicializo el acumulador
for i in range(n):
        sum = sum + i # Cada vez que ocurre la condicion le agrego uno
print 'Hasta el numero ',n,', la sumatoria es: ',sum
```

La operacioń que estamos realizando es igual a la del contador, y también podemos escribirla en forma pythonica como:

sum = sum + i $\rightarrow$ sum + = i

Ejercicio: Secuencia de Fibonacci

El próximo número en la serie de Fibonacci es la suma de los dos últimos, comenzando por 0 y 1.

Matematicamente esta serie es infinita. Pero por supuesto en el programa vamos a tener que limitar el número de ciclos.
¿ Como detenemos el proceso? O después de un numńero fijo de ciclos o cuando el término de la serie llega a un valor máximo deseado.

Respuesta: Secuencia de Fibonacci

```
nciclos = 10
num1 = 0
num2 = 1

for i in range(nciclos):
    print(num2, end=" ")
    num1,num2 = num2, num1+num2
```

```
1 1 2 3 5 8 13 21 34 55
```

```
ntot = 100
num1 = 0
num2 = 1

while num2 < ntot:
    print(num2, end=" ")
    num1,num2 = num2, num1+num2
```

```
1 1 2 3 5 8 13 21 34 55 89
```

Notar el uso de la tupla. Pythonico!

Evita el uso de una tercera variable para intercambiar valores:

```
a,b = b, a
```

```
temp=a
a=b
b=temp
```

Puede utilizarse para asignar múltiples variables a la vez:

```
pi,tk,g = 3.14,-273.15,9.80
```

Bucles anidados. Ejemplo

Queremos que un estudiante de la primaria, Manuel mi hijo, practique las tablas de multiplicación.

```
ierror=0
for i  in range(1,10):
    for j in range(1,10):
        cadena='Cuanto es:  '+str(i)+'x'+str(j)+' ? '
         res=int ( input(cadena) )
         if (res != i*j):
            ierror+=1
            print 'Has cometido ',ierror,' errores'

if  (ierror < 3):
    print 'Te felicito. Podes ir a jugar'
else:
    print 'Te quedaste sin futbol.'
```

Enumerate

Hay veces que ademas del ciclado en la lista necesitamos tener un índice del número de ciclo:

```
enumerate(secuencia, start=0)
```

Esta función da dos salidas, el índice de la secuencia y su elemento.

Supongamos que a una lista de flotantes queremos multiplicar/adicionar el cuadrado de su ubicación:

```
nros=[5,10,21,57]
res=[]
sum=0
for i,nro in enumerate(nros):
    sum+=nro * i**2
    res.append(nro+i**2)
```

Pythonico

```
nros=[5,10,21,57]
res=[]
sum=0
for i in range(nros):
    sum+=nros[i] * i**2
    res.append(nro[i]+i**2)
```

No tan lindo.

El enumerate es un caso particular de los generadores que veremos en la próxima clase.

Transformación de un número binario a decimal

Queremos transformar con un numero binario ingresado y controlando la entrada.

$$n_d = \sum_{i=0}^{n-1} \text{dig\_bin}\, 2^i$$