Temario

- Variables lógicas. True False.
- Bifurcaciones. If else.
- Loops. Ciclos. while for.
- Funciones. def.
- Interfaces de ventanas.

Variables lógicas

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

```
>>> lpreg=True
```

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

'bool'=boolean

Operaciones con variables lógicas

Existen tres operaciones de variables lógicas: and, or y not.

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

Operador and

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

True and True $\rightarrow$ True

True and False $\rightarrow$ False

False and False $\rightarrow$ False

Operador or

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

True or True $\rightarrow$ True

True or False $\rightarrow$ True

False or False $\rightarrow$ 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? ==

```
>>> 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

Operador para cadena de caracteres: strings

```
>>> 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 (en muchos contextos es similar a ==, pero mas pythonic porque es legigle).

```
>>> x0 is 5
```

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

Las variables las puedo definir como 'None'

La instrucción if

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'
...     quit()
```

La estructura de la instrucción if es:

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

(4 espacios en blanco) Haga esto

Los espacios en blanco, tabulación, son parte de la instrucción.

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=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'
```

Varias opciones elif.

Hay veces que necesitamos varias opciones no solo dos. Para esto existe el elif. Es una mezcla de else y de if, de lo contrario si 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=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 ningun opcion válida”.

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=1
while i<=8:
    print i
    i=i+1
```

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

Otra alternativa 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 i in a:
...     print i,')'
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 )
```

Bucles con for

El for es similar al while pero mucho mas compacto! bien pythonic!

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 cortar un ciclo de repeticiones 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 >
```

Bucles anidados. Ejemplo

Queremos que un estudiante de la primaria, Manuel mi hijo, estudie 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=input(cadena)
         if (res != i*j):
            ierror+=1
            print 'Has cometido ',ierror,' errores'
            if (ierror >= 3):
                print 'Te quedaste sin futbol.'
                break
    if  (ierror >= 3): # aqui rompo con el 2do bucle anidado
        break
if  (ierror < 3):
    print 'Te felicito. Podes ir a jugar'
```

Funciones (mas alla de las matemáticas)

Que sucede si queremos realizar el mismo conjunto de instrucciones en numerosas partes de un programa? tenemos que repetir estas instrucciones en todas partes?

Una de las herramientas mas útiles de python son las funciones, que actuan de una forma muy similar a las funciones matemáticas, las funciones tienen un conjunto de variables de entrada, y tienen una o varias variables de salida/resultados.

Las funciones se definen con un def luego el nombre de la función y entre paréntesis los argumentos de entrada:

```
def funcion_3Dgral(x,y,z)
```

Todas las instrucciones de la función van tabuladas a 4 espacios. Y cuando queremos que el flujo vuelva al programa principal con el resultado de la función hacemos,

```
return resultado1,resultado2
```

Se aconseja despues del def, un comentario explicando que es lo que hace la función, las variables de entrada y las de salida.

Funciones (mas alla de las matemáticas)

Supongamos que en nuestro programa tenemos que hacer cambios de grados Celsius a Fahrenheit, sabemos que la función es $F(C) = \frac{9}{5}C + 32$,

```
def trans_c2f(tcel):
    " Transforma temperatura de Celsius a Fahrenheit"
    tfah=9./5.*tcel+32.0
    return tfah
```

Luego en el programa principal llamamos a la función:

```
ta = 10
F1 = trans_c2f(ta)
temp = trans_c2f((15.5)
print trans_c2f(a+1)
sum_temp = trans_c2f(10.0) + trans_c2f(20.0)
```

Aun cuando la función se utilice una sola vez en el programa principal, conceptualmente un programa es mucho mas claro cuando cada “función” esta definida en una “función independiente”.

Funciones. Todo funciones.

REGLA DE ORO: programa principales pequeños que llaman a funciones.

Las funciones son unidades básicas independientes, esto nos permite reciclarlas, ej. en cualquier programa que necesite transformar temperatura puedo utilizar la función trans_c2f.

Además tiene muy bien definido todo lo que necesita para funcionar (Argumentos de entrada) y cuales son los resultados (variables de salidas).

Variables locales. Nacen y mueren en la función

Por otro lado todas las variables que se definan en una función son variables locales, es decir se crean para la función, pero luego en el return se destruyen y no afectan el resto del programa.

```
def trans_c2f(tcel):
    factor=9./5.
    ta=factor*tcel+32.0
    return ta
ta = 10
F1 = trans_c2f(ta)
print 'Variable ta: ',ta
print 'Variable factor: ',factor
```

Que da el print de ta? el valor que se calcula en la función o el que esta en el programa principal?

Rta: 10 (el del programa principal, la función no cambia el valor de ta).

Que da el print del factor?

Argumentos múltiples. Valores por default

```
def posrel(t,v0,g=9.81)
    y=v0*t- 0.5*g*t**2
    return y
```

A esta función la puedo llamar como:

```
y=posrel(10,5)
```

para todos los casos que este en la Tierra.

Si quiero calcular la posición en Marte

```
y=posrel(10,5,g=3.711)
```

Si llama a la función sin ese argumento tomara el valor por default. Si la llamo con el argumento puedo ponerle el valor que quiero.

Notar que como es opcional le pongo el nombre del argumento.

Switches como argumentos

En el problema de caida libre queremos tener una opción para que imprima el resultado en la funcion:

```
def posrel(t,v0,g=9.81,salida=None):
    y=v0*t- 0.5*g*t**2
    if salida is not None:
        print 'La posicion es: ',y
    return y
```

Si la variable salida esta defina que imprima el resultado (esto actua de switch, pero tambien podria tener valores).

Esta función se puede llamar como:

```
posrel(10,5,salida=1)
posrel(10,5,g=3.711)
posrel(10,5,g=3.711,salida=1)
```

Donde ubicamos a las funciones?

Si estamos escribiendo en un solo archivo al programa principal y las funciones:

Las funciones deben ir antes del llamado a éstas.

```
# Primero Funcion
def trans_c2f(tcel):
    factor=9./5.
    ta=factor*tcel+32.0
    return ta
# Llamando a la funcion desde el programa principal
F1 = trans_c2f(10.)
```

Esto es asi porque python es un interprete, entonces necesita tener “interpretada” la función, antes que se la quiera utilizar (de lo contrario no sabría lo que es “trans_c2f”.

Donde ubicamos a las funciones?. Librerías

Las funciones pueden ir en un archivo aparte en el mismo directorio: una librería

Supongamos que ubicamos en el archivo transformaciones.py las funciones trans_c2f y trans_f2c. En el programa principal transforma.py hacemos:

```
#!/usr/bin/env python
"Transforma de grados centigrados a fahrenheit y viceversa"
import transformaciones as tr

print 'Que desea calcular: '
opt=input('(1) Celsius a Fahrenheit, (2) Fahrenheit a Celsius:')
if opt == 1:
    tcel=input('Ingrese Temperatura en Celsius')
    tfah=tr.trans_c2f(tcel)
    print 'Temperatura de ',tcel,' Celsius es ',tfah,' Fahrenheit'
elif opt == 2:
    tfah=input('Ingrese temperatura en Fahrenheit')
    tcel=tr.trans_f2c(tfah)
    print 'Temperatura de ',tfah,' Fahrenheit es ',tcel,' Celsius'
```

Librerías en carpetas python

Las funciones pueden ir en un archivo en otro directorio especial. Supongamos que tenemos muchas funciones en archivos y entonces hacemos una carpeta, “funciones” y ponemos todos los archivos “.py” alli (Excepto el programa principal).

Esta es la forma en la cual estan escritas las librerías “numpy”, “os” etc.

Para que el “funciones” funcione como un directorio python, debemos poner un archivo adentro __init__.py (solo contiene un espacio “ ” por el momento)

Poner el archivo transformaciones.py en la carpeta funciones

```
$ mv transformaciones funciones
```

Luego en el programa principal transforma.py lo único que hacemos es:

```
import funciones.transformaciones as tr
```

El resto igual.

Interfaces gráficas

En general todas las aplicaciones de software contienen interfaces gráficas que interactuan con el usuario. Python tiene grandes funcionalidades a través de librerías para la creación y el manejo de estas interfaces.

Es muy simple aprender a utilizar las herramientas mínimas y a partir de éstas se pueden construir todo un software!

Los widgets son todo un mundo. Veamos los elementos principales como para despertarles la curiosidad.

```
from Tkinter import *
ppal = Tk()
entradaC = Entry(ppal, width=4)
entradaC.pack(side='left') #principal

unidadC = Label(ppal, text='Celsius') #ventana celsius
unidadC.pack(side='left')

calcula = Button(ppal, text=' corresponde a ', command=compute)
calcula.pack(side='left', padx=4)

resultadoF = Label(ppal, width=4) #ventana con el resultado.
resultadoF.pack(side='left')

unidadF = Label(ppal, text='Fahrenheit') #ventana de fahrenheit
unidadF.pack(side='left')

ppal.mainloop()
```

Interfaces gráficas

```
def compute():
    C = float(entradaC.get())
    F = (9./5)*C + 32
    resultadoF.configure(text='%g' % F)
```

En aplicaciones en serio, a las variables que representan estructuras gráficas conviene agregarles un identificador en el nombre: por ejemplo wgt (Widget) resultadoF_wgt, calcula_wgt, etc.