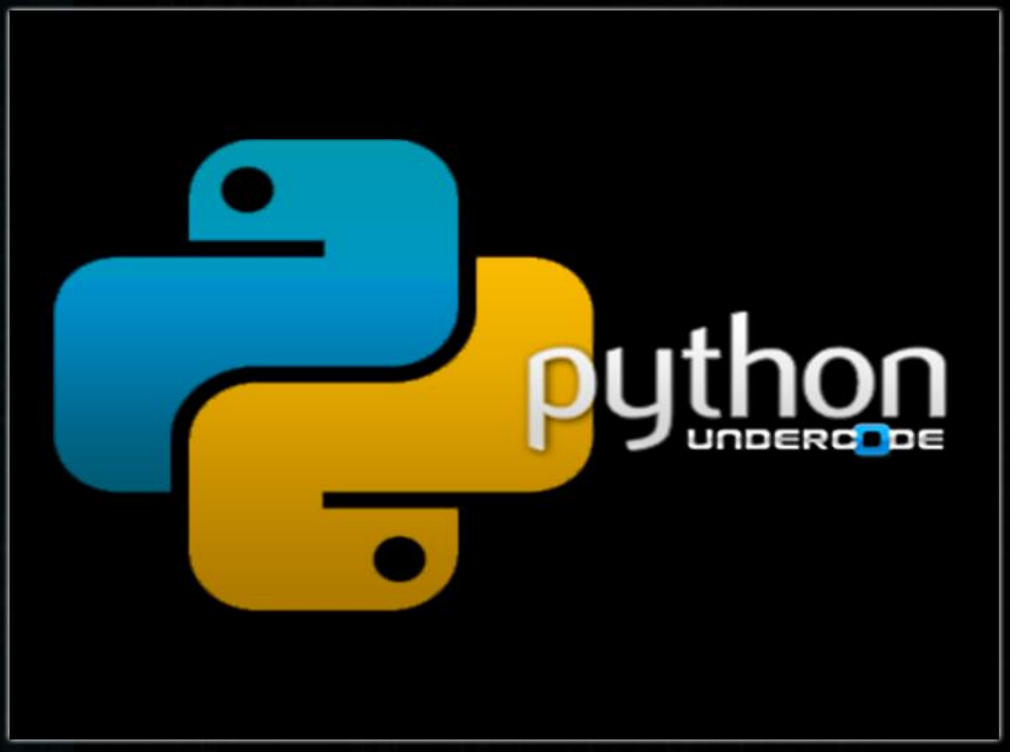
#2

UNDERCODE

TALLER DE PYTHON



TEMAS

- ESTRUCTURAS DE CONTROL
- VARIABLES
- ENTRADA DE DATOS
- SANGRADO
- CONDICIONALES
- BUCLES
- Y MUCHO MÁS..!

TUTOR

11SEP

Estructuras de control:

¿Qué son las estructuras de control?

Las estructuras de control nos permiten modificar el flujo del programa. Piensa en el programa como una piedra que baja por un tubo, entra por una parte y sale por la otra, siempre sigue el mismo camino. Ahora piensa en el mismo tubo pero con otros tubos incrustados a los costados, ahora cuando lanzas la piedra, esta puede entrar por alguno de los tubos que están a los costados haciendo que la piedra regrese al comienzo, valla al final o simplemente a otra parte del tubo; ¡Modificamos su camino! Eso es justo lo que logramos con las estructuras de control.

Antes de continuar con este tema, vamos a aclarar un tema de la primer entrega del taller para poder continuar sin problemas.

Las variables:

Las variables son espacios en memoria donde podemos guardar datos para usarlos luego. Pueden ser números, cadenas de caracteres o valores booleanos (True o False).

Python se caracteriza por ser de tipado fuerte y dinámico.

Fuertemente tipado:

Significa que no puedes operar con dos variables de diferentes tipos como si estas fuesen iguales (tema visto en la primer entrega de este taller). Sin embargo Python cuenta con funciones para conocer y convertir el tipo de las variables que lo veremos más adelante en el taller.

Tipado dinámico:

Significa que no tienes que declarar la variable y su tipo antes de usarla (como sucede en Visual Basic), Python se encarga de ello internamente cada vez que se le asigna un valor a una variable.

Variables booleanas:

Las variables booleanas son las que pueden guardar dos valores: True o False (verdadero o falso) y aunque a simple vista parecen inútiles, las necesitaremos mucho de ahora en adelante para los condicionales y los bucles.

Operadores booleanos:

and: Devuelve **True** sólo si todos los operandos son **True**

or: Devuelve **True** si alguno de los operandos es **True**

not: Devuelve el inverso del valor (si es **True** devuelve **False** y viceversa)

Esto nos servirá más adelante cuando tengamos que hacer condiciones relativamente más complejas.

Ejemplo and:

```
Python 2.7.3 (default, Dec 22 2012, 21:14:12)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more informa
>>> 1==1 and 1==1
True
```

Tenemos dos expresiones a evaluar, en este caso las mismas a ambos lados (1==1) como sabemos que uno es igual a uno a ambos lados del and tenemos **True**. Así que la expresión quedaría de la siguiente forma:

True and True

Por lo que el valor que nos arrojará Python es **True**.

Ejemplo or:

```
>>> 1==1 or 8<4
True
>>>
```

En este caso tenemos 1==1 a la izquierda y 8<4 a la derecha. Como sabemos, uno es igual a uno así que a la izquierda tenemos un **True** pero en la derecha tenemos 8<4 y como en ningún universo paralelo esa expresión sería verdadera, a la derecha tenemos un **False**. Por lo que la expresión nos quedaría:

True or False

Y como ya vimos más arriba que si uno de los operandos es **True** el resultado será **True,** Python nos devuelve **True**

Ejemplo not:

```
>>> not 5==5
False
>>> _
```

Este casi no necesita explicación, cinco es igual a cinco, así que la expresión nos quedaría:

not True

Así que Python nos devuelve **False**

Las asignaciones:

Ahora que sabemos que son las variables y para que nos sirven, necesitamos saber como darle valores a las variables.

En Python para asignar un valor a una variable usamos **=** y la sintaxis es:

variable = valor

Donde **variabale** es el nombre de nuestra variable y **valor** el valor que le asignaremos a la variable.

Debemos tener en cuenta una cosa: si la variable no existe, Python la crea y le asigna el valor, pero si la variable **ya existe** Python le asignará a la variable el último valor.

```
Python 2.7.3 (default, Dec 22 2012, 21:14:12)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> variable = "Taller Python Underc0de"
>>> print variable
Taller Python Underc0de
>>> type(variable)
<type 'str'>
>>> variable = 1223
>>> print variable
1223
>>> type(variable)
<type 'int'>
>>> _
```

Revisemos el ejemplo: creamos la variable **variable** y le asignamos el valor “Taller Python Underc0de” y si hacemos un **type(variable)** podemos ver que el tipo de la variable es **str** (Cadena de caracteres). Ahora a la misma variable le asignamos el valor: 1223. Vemos que al hacer un print nos muestra 1223 (El último valor que le asignamos a la variable). Pero además si hacemos un **type(variable)** vemos que el tipo de la variable es **int**

(entero). Sé lo que estas pensando y sí, es gracias al tipado dinámico.

Supongamos que tenemos un valor numérico en una variable y necesitas aplicar una operación sobre ella y guardar de nuevo el resultado en la misma variable. Tenemos dos opciones:

```
Python 2.7.3 (default, Dec 22 2012, 21:14:12)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> valor = 5
>>> valor = valor + 5
>>> print valor
10
>>> _
```

Esta es la primer opción, lo que hacemos es llevarle a variable **valor** su mismo valor sumado 5.

```
Python 2.7.3 (default, Dec 22 2012, 21:14:12)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> valor = 5
>>> valor += 5
>>> print valor
10
>>>
```

Esta es nuestra segunda opción, obtenemos el mismo resultado y nos ahorramos un par de caracteres.

Además de **+=** podemos usar:

Operador	Acción
+=	Suma o concatena
-=	Resta
*=	Multiplica o concatena
/=	Divide
%=	Resto de la división
**=	Eleva a una potencia

Ejemplos concatenación:

```
Python 2.7.3 (default, Dec 22 2012, 21:14:12)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> valor = "Taller"
>>> valor += " Underc0de"
>>> print valor
Taller Underc0de
>>> _
```

Concatenamos los valores “Taller” y “Underc0de” asignandolos a la misma variable

```
Python 2.7.3 (default, Dec 22 2012, 21:14:12)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> valor = "Underc0de "
>>> valor *= 3
>>> print valor
Underc0de Underc0de Underc0de
>>> _
```

Multiplicamos el valor “Underc0de” 3 veces

Nota: En Python si tratamos de multiplicar una cadena de caracteres el resultado será la misma cadena repetida la misma cantidad de veces por la que la multiplicamos. Como pudimos ver en el ejemplo de arriba.

Nota: La función **type()** nos devuelve el tipo de una variable.

Convertir variables:

Python nos brinda un par de funciones con las cuales podemos convertir los tipos de las variables. Así si tienes una variable de tipo **int** y necesitas un **str** puedes convertir la variable.

Función	Convierte a:
Str()	Cadena de caracteres
Int()	Entero
Float()	Flotante

Su uso es simple, simplemente debes poner la variable que quieres convertir entre los paréntesis de la función:

```
>>> valor = "123"
>>> print int(valor)
123
>>> print int(valor) + 123
246
>>> valor = 153
>>> print str(valor)
153
>>> print str(valor) + "153"
153153
>>> valor = 183
>>> print float(valor)
183.0
>>> print float(valor) + 100
283.0
>>> _
```

Ingreso de datos por parte del usuario:

¿Qué sería de un programa sin interacción con el usuario?

En Python contamos con dos funciones que permiten al usuario ingresar datos al script. Ellas son: **input()** y **raw_input()**

Input(): Permite al usuario ingresar valores numéricos.

raw_input(): Permite al usuario ingresar cadenas.

Sintaxis:

variable = **input**(“Texto que se mostrará al usuario”)

variable es la variable donde se guardarán los datos ingresados por el usuarios, lo que está entre comillas es el texto que se le mostrará al usuario.

Nota: Ambas funciones cuentan con la misma sintaxis.

Ejemplo:

```
#!/usr/bin/python2
#-*- coding:utf-8 -*-

Datos = raw_input("Ingrese el usuario: ")
print "El usuario es", Datos

Numero = input("Ingrese el número a multiplicar pór 5: ")
print Numero * 5
```

Con las primeras dos líneas pedimos al usuario que “Ingrese el usuario”, lo guardamos en la variable Datos y luego lo mostramos en la pantalla.

Con las últimas dos líneas, pedimos al usuario un número para multiplicarlo por cinco y luego mostrar el resultado de la operación.

Si ejecutamos el script, tenemos algo como esto:

```
Ingrese el usuario: Once
El usuario es Once
Ingrese el número a multiplicar pór 5: 10
50
```

El Sangrado (Identación):

Python no usa llaves o palabras clave para señalar el final de un bloque de código (en JS se usan los corchetes o en Visual Basic se usa la palabra reservada **end).** Python usa el sangrado/tabulación/identación, todo lo que este tabulado es un bloque de código para Python que termina donde la tabulación lo hace.

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

def Suma(num1, num2):
    print num1 + num2

Suma(1, 5)
```

(No importa si no entiendes este código, lo harás más adelante, por el momento sólo nos sirve como un ejemplo.)

En el código anterior definimos la función **Suma** que se encarga de sumar dos valores y mostrar el resultado en la pantalla (puedes ejecutar el script y mirar el resultado). Los dos puntos nos indica que lo que sigue es un bloque de código, por ende lo que queremos que valla dentro de la función va tabulado.

¿Dónde termina la función?

La función termina donde termina la tabulación, en este caso en el print. También aplica para los condicionales y bucles. Es fundamental tener claro esto antes de poder continuar.

Condicionales:

Los condicionales son instrucciones que evalúan una condición, si el resultado de la condición da como resultado un valor **True** (verdadero) ejecuta un bloque de código, de lo contrario tenemos dos opciones, seguir con el rumbo del programa (**sin ejecutar el código dentro del condicional**) o evaluar otra condición (**elif**). Podemos evaluar tantas condiciones como deseemos.

El condicional if:

Evalúa una condición, si el resultado es True ejecuta el código que viene identado después del **if**.

Sintaxis:

if (condición):

    Código a ejecutar

Ejemplo:

Queremos que nuestro programa nos muestre en pantalla un "si" sí un número es mayor que otro.

```
#!/usr/bin/python
# -*- coding:utf-8 -*-

Numero1 = 7
Numero2 = 6

if Numero1 > Numero2:
    print "Si"
```

Si ejecutamos el código, vemos que nos muestra un "Si" ya que 7 es mayor que 6.

El condicional elif:

Si la condición del primer **if** es False (falsa), **elif** nos permite evaluar otra condición. Podemos usar tantos **elif** como deseemos.

Sintaxis:

if (condición):

    Código a ejecutar

elif (condición):

    Código a ejecutar

Ejemplo:

Queremos modificar el código anterior para que en caso de que el primer número sea menor que el segundo, imprima en pantalla un “No”

```
#!/usr/bin/python
# -*- coding:utf-8 -*-

Numero1 = 7
Numero2 = 6

if Numero1 > Numero2:
    print "Si"
elif Numero1 < Numero2:
    print "No"
```

Si jugamos con el valor de las variables **Numero1** y **Numero2** podemos ver los distintos resultados que nos arroja el script.

El condicional else:

En el caso de que los condicionales **if** y **elif** resulten **False** todavía tenemos una última oportunidad para evitar que el programa retome su rumbo habitual (siga bajando por el tubo). En este caso ya no se evalúa una condición, una vez llegado a este punto se ejecuta el código identado después del **else.**

Sintaxis:

if (condición):

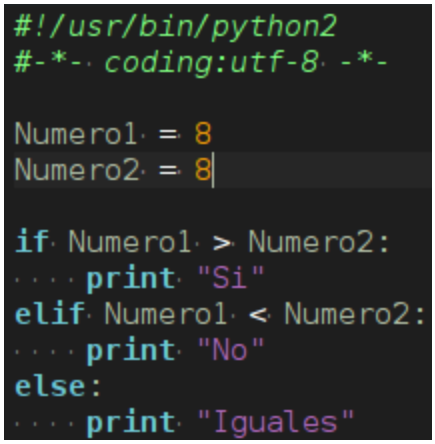
    Código a ejecutar

else:

Código a ejecutar

Ejemplo:

Ahora queremos modificar el script anterior para que nos imprima en la pantalla un “Iguales” en el caso de que las variables tengan el mismo valor.



```
#!/usr/bin/python2
#-*- coding:utf-8 -*-

Numero1 = 8
Numero2 = 8

if Numero1 > Numero2:
    print "Si"
elif Numero1 < Numero2:
    print "No"
else:
    print "Iguales"
```

En este caso no necesitamos evaluar una condición ya que estamos comparando dos números y sólo hay tres posibilidades, que el primero sea mayor que el segundo, que el primero sea menor que el segundo, que ambos sean iguales. Ya con el **if** y con el **elif** tenemos cubiertas las dos primeras posibilidades así que sólo queda una posibilidad y es la que capturamos con el **else.**

Operadores de comparación:

Los operadores de comparación sirven para comparar (valga la redundancia) dos valores y devuelve True o False según el caso. Los operadores de comparación que usamos en Python son:

Operador	Significado
!=	Distinto de
==	Igual a
<	Menor que
<=	Menor igual que
>	Mayor que
>=	Mayor igual que

Nota: En Python se usa = para asignar valores y == para hacer comparaciones.

Estos son los operadores que podemos usar cuando necesitamos evaluar una condición.

```
Python 2.7.3 (default, Dec 22 2012, 21:14:12)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> "123"==123
False
>>> 1!=2
True
>>> 5>5
False
>>> 9<15
True
>>> 8>=8
True
>>> 1<=2
True
>>>
```

Bucles:

Los bucles son partes del código que se ejecuta una cantidad determinada de veces, ya sea conocida por el programador o no. Por ejemplo, si tienes que mostrar mil números es más fácil ejecutar una misma porción de código mil veces que escribir mil print a mano.

Los bucles que podemos usar en Python son: **for** y **while.**

Bucle for:

Usamos este bucle cuando sabemos cuántas veces queremos que se repita, también podemos recorrer listas y tuplas (las veremos más adelante).

Sintaxis:

for variable in range(rango):

    bloque de código a ejecutar

Variable nos sirve de contador y es la que va a contener la "posición" donde está el bucle o el elemento de la lista o tupla.

in range (rango**)** define la cantidad de veces que se va a repetir, teniendo en cuenta que en contador comienza en cero. Si queremos que el contador comience en un valor distinto de cero, podemos hacerlo de esta manera:

for variable in range(valor_inicial, valor_final):

Bloque de código a ejecutar

Ejemplo:

Imprimir los números del 0 al 10.

```
#!/usr/bin/python2
#-*- coding:utf-8 -*-

for i in range(11):
    print i
```

Bucle while:

Este bucle se repite siempre y cuando se cumpla una condición. Es acá donde debemos tener cuidado, **si la condición siempre es verdadera**, el bucle se ejecutará infinitamente.

Sintaxis:

while condición:

Código a ejecutar

Ejemplo:

Mostrar los números del 0 al 10.

```
#!/usr/bin/python2
#-*- coding:utf-8 -*-

Contador = 0
while Contador <= 10:
    print Contador
    Contador += 1
```

En el caso del **while** si necesitamos un contador, lo debemos declarar nosotros; En este caso la variable que nos sirve de contador es **Contador**

En este ejemplo mientras la variable **Contador** sea menor o igual a 10, se va a ejecutar el bloque de código del **while**. Imprime el valor de **Contador** y suma una unidad al valor actual de la variable **Contador.** Esta es la parte más importante ya que si no incrementamos el valor de la variable entraríamos a un bucle infinito ya que **0 siempre será menor que 10**.

Ahora, veamos otro ejemplo tratando de tomar todo lo ya visto:

```
#!/usr/bin/python2
#-*- coding:utf-8 -*-

while True:
    Cnd = raw_input("Ingrese una palabra (salir para salir): ")
    if Cnd == "salir":
        print "Adios"
        break
    else:
        print "Usted ha ingresado: " + Cnd
```

En este ejemplo vemos dos cosas nuevas:

1. **while True:** Hace un bucle infinito ya que en lugar de una condición a evaluar le pasamos directamente **True.**

2. **break:** Como estamos en un bucle infinito necesitamos una forma de salir de él y para eso sirve **break,** nos permite salir del bucle para que el programa tome su camino normal.

Ahora que tenemos todo claro vamos a analizar el código:

while True: #Ingresamos en un bucle infinito

Cnd = raw_input() #pedimos al usuario que ingrese una cadena

if Cnd == "salir": #Si lo que el usuario ingresó fue "salir"

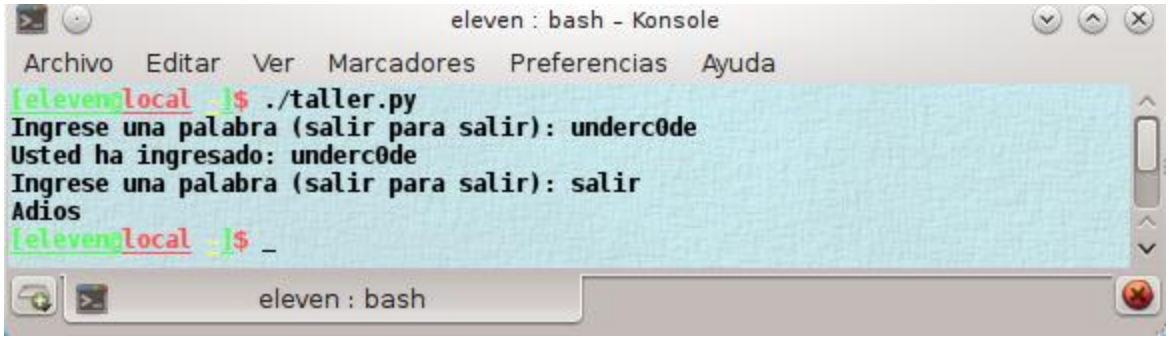
print "Adios" #imprimimos "Adios"

break #Salimos del bucle

else: #Si el usuario NO ingresó "Salir"

print "blablabla" + Cnd #Imprimimos lo mismo que el usuario ingresó. Una vez llegados a este punto el proceso se repite hasta que el usuario ingrese "salir"

Y si lo ejecutamos:



Ejercicios:

Dar el resultado de las siguientes expresiones y como quedarían:

A. “asd” == “asd” and 1>5

B. 1*4==5 or **False**

C. not **True**

Escribir un script usando el **bucle for** que imprima 20 veces la palabra “Underc0de”

Escribir un script utilizando el **bucle while** que muestre el resultado de la suma de los números del 0 al 10.

Escribir un script que le pida al usuario un número y le diga si el número es mayor que 10 y menor que 15.

Encuentra el error:

```
#!/usr/bin/python2
#-*- coding:utf-8 -*-

Contador = 0

while Contador < 10:
    print "Hola, esta es la vuelta", Contador
    Contador -= 1
```