02 Estructura y elementos del lenguaje

Dentro de los **lenguajes informáticos**, Python, pertenece al grupo de los **lenguajes de programación** y puede ser clasificado como un **lenguaje interpretado**, **de alto nivel**, **multiplataforma**, de **tipado dinámico** y **multiparadigma**. A diferencia de la mayoría de los lenguajes de programación, **Python nos provee de reglas de estilos**, a fin de poder escribir código fuente más legible y de manera estandarizada. Estas reglas de estilo, son definidas a través de la ***Python Enhancement Proposal* Nº 8 (PEP 8)** , la cual iremos viendo a lo largo del curso.

Comparte el conocimiento: Creative Commons Atribución-NoComercial-CompartirIgual 3.0

GLOSARIO

Lenguaje informático:es un idioma artificial, utilizado por ordenadores, cuyo fin es transmitir información de algo a alguien. Los lenguajes informáticos, pueden clasificarse en: a) lenguajes de programación (Python, PHP, Pearl, C, etc.); b) lenguajes de especificación (UML); c) lenguajes de consulta (SQL); d) lenguajes de marcas (HTML, XML); e) lenguajes de transformación (XSLT); f) protocolos de comunicaciones (HTTP, FTP); entre otros.

Lenguaje de programación: es un lenguaje informático, diseñado para expresar órdenes e instrucciones precisas, que deben ser llevadas a cabo por una computadora. El mismo puede utilizarse para crear programas que controlen el comportamiento físico o lógico de un ordenador. Está compuesto por una serie de símbolos, reglas sintácticas y semánticas que definen la estructura del lenguaje.

Lenguajes de alto nivel: son aquellos cuya característica principal, consiste en una estructura sintáctica y semántica legible, acorde a las capacidades cognitivas humanas. A diferencia de los lenguajes de bajo nivel, son independientes de la arquitectura del hardware, motivo por el cual, asumen mayor portabilidad.

Lenguajes interpretados: a diferencia de los compilados, no requieren de un compilador para ser ejecutados sino de un intérprete. Un intérprete, actúa de manera casi idéntica a un compilador, con la salvedad de que ejecuta el programa directamente, sin necesidad de generar previamente un ejecutable. Ejemplo de lenguajes de programación interpretado son Python, PHP, Ruby, Lisp, entre otros.

Tipado dinámico: un lenguaje de tipado dinámico es aquel cuyas variables, no requieren ser definidas asignando su tipo de datos, sino que éste, se auto-asigna en tiempo de ejecución, según el valor declarado.

Multiplataforma: significa que puede ser interpretado en diversos Sistemas Operativos como GNU/Linux, Windows, Mac OS, Solaris, entre otros.

Multiparadigma: acepta diferentes paradigmas (técnicas) de programación, tales como la orientación a objetos, aspectos, la programación imperativa y funcional.

Código fuente: es un conjunto de instrucciones y órdenes lógicas, compuestos de algoritmos que se encuentran escritos en un determinado lenguaje de programación, las cuales deben ser interpretadas o compiladas, para permitir la ejecución del programa informático.

Comparte el conocimiento: Creative Commons Atribución-NoComercial-CompartirIgual 3.0

Elementos del Lenguaje

Como en la mayoría de los lenguajes de programación de alto nivel, en Python se compone de una serie de elementos que alimentan su estructura. Entre ellos, podremos encontrar los siguientes:

Variables

Una variable es un espacio para almacenar datos modificables, en la memoria de un ordenador. En Python, una variable se define con la sintaxis:

```
nombre_de_la_variable = valor_de_la_variable
```

Cada variable, tiene un nombre y un valor, el cual define a la vez, el tipo de datos de la variable.

Existe un tipo de "variable", denominada **constante**, la cual se utiliza para definir valores fijos, que no requieran ser modificados.

PEP 8: variables
Utilizar nombres descriptivos y en minúsculas. Para nombres compuestos, separar las palabras por guiones bajos. Antes y después del signo =, debe haber uno (y solo un) espacio en blanco

Correcto: `mi_variable = 12`
Incorrecto: `MiVariable = 12 | mivariable = 12 | mi_variable=12 | mi_variable = 12`

PEP 8: constantes
Utilizar nombres descriptivos y en mayúsculas separando palabras por guiones bajos.

Ejemplo: `MI_CONSTANTE = 12`

Para **imprimir un valor en pantalla**, en Python, se utiliza la palabra clave `print`:

```
mi_variable = 15
print mi_variable
```

Lo anterior, imprimirá el valor de la variable mi_variable en pantalla.

Comparte el conocimiento: Creative Commons Atribución-NoComercial-CompartirIgual 3.0

Tipos de datos

Una variable (o constante) puede contener valores de diversos tipos. Entre ellos:

Cadena de texto (string):

```
mi_cadena = "Hola Mundo!"

mi_cadena_multilinea = """
Esta es una cadena
de varias lineas
"""
```

Número entero:

```
edad = 35
```

Número entero octal:

```
edad = 043
```

Número entero hexadecimal:

```
edad = 0x23
```

Número real:

```
precio = 7435.28
```

Booleano (verdadero / Falso):

```
verdadero = True
falso = False
```

Existen además, otros tipos de datos más complejos, que veremos más adelante.

Operadores Aritméticos

Entre los operadores aritméticos que Python utiliza, podemos encontrar los siguientes:

Símbolo	Significado	Ejemplo	Resultado
+	Suma	a = 10 + 5	a es 15
-	Resta	a = 12 - 7	a es 5
-	Negación	a = -5	a es -5
*	Multiplicación	a = 7 * 5	a es 35
**	Exponente	a = 2 ** 3	a es 8
/	División	a = 12.5 / 2	a es 6.25
//	División entera	a = 12.5 / 2	a es 6.0
%	Módulo	a = 27 % 4	a es 3

> **PEP 8: operadores**
> Siempre colocar un espacio en blanco, antes y después de un operador

Un ejemplo sencillo con variables y operadores aritméticos:

```
monto_bruto = 175
tasa_interes = 12
monto_interes = monto_bruto * tasa_interes / 100
tasa_bonificacion = 5
importe_bonificacion = monto_bruto * tasa_bonificacion / 100
monto_neto = (monto_bruto - importe_bonificacion) + monto_interes
```


Comparte el conocimiento: Creative Commons Atribución-NoComercial-CompartirIgual 3.0

Comentarios

Un archivo, no solo puede contener código fuente. También puede incluir comentarios (notas que como programadores, indicamos en el código para poder comprenderlo mejor).

Los comentarios pueden ser de dos tipos: de una sola línea o multi-línea y se expresan de la siguiente manera:

```
# Esto es un comentario de una sola línea
mi_variable = 15

"""Y este es un comentario
de varias líneas"""
mi_variable = 15

mi_variable = 15  # Este comentario es de una línea también
```

En los comentarios, pueden incluirse palabras que nos ayuden a identificar además, el subtipo de comentario:

```
# TODO esto es algo por hacer
# FIXME esto es algo que debe corregirse
# XXX esto también, es algo que debe corregirse
```

PEP 8: comentarios
Comentarios en la misma línea del código deben separarse con dos espacios en blanco. Luego del símbolo # debe ir un solo espacio en blanco.

Correcto:
```
a = 15  # Edad de María
```
Incorrecto:
```
a = 15 # Edad de María
```

Tipos de datos complejos

Python, posee además de los tipos ya vistos, 3 tipos más complejos, que admiten una **colección de datos**. Estos tipos son:

- Tuplas
- Listas
- Diccionarios

Estos tres tipos, pueden almacenar colecciones de datos de diversos tipos y se diferencian por su sintaxis y por la forma en la cual los datos pueden ser manipulados.

Tuplas

Una tupla **es una variable que permite almacenar varios datos inmutables** (no pueden ser modificados una vez creados) de tipos diferentes:

```
mi_tupla = ('cadena de texto', 15, 2.8, 'otro dato', 25)
```

Se puede acceder a cada uno de los datos mediante su índice correspondiente, siendo 0 (cero), el índice del primer elemento:

```
print mi_tupla[1]  # Salida: 15
```

También se puede acceder a una porción de la tupla, indicando (opcionalmente) desde el índice de inicio hasta el índice de fin:

```
print mi_tupla[1:4]  # Devuelve: (15, 2.8, 'otro dato')
print mi_tupla[3:]   # Devuelve: ('otro dato', 25)
print mi_tupla[:2]   # Devuelve: ('cadena de texto', 15)
```

Otra forma de acceder a la tupla de forma inversa (de atrás hacia adelante), es colocando un índice negativo:

```
print mi_tupla[-1]  # Salida: 25
print mi_tupla[-2]  # Salida: otro dato
```

Listas

Una lista es similar a una tupla con la diferencia fundamental de que permite modificar los datos una vez creados

Comparte el conocimiento: Creative Commons Atribución-NoComercial-CompartirIgual 3.0

```
mi_lista = ['cadena de texto', 15, 2.8, 'otro dato', 25]
```

A las listas se accede igual que a las tuplas, por su número de índice:

```
print mi_lista[1]    # Salida: 15
print mi_lista[1:4]  # Devuelve: [15, 2.8, 'otro dato']
print mi_lista[-2]   # Salida: otro dato
```

Las lista NO son inmutables: permiten modificar los datos una vez creados:

```
mi_lista[2] = 3.8  # el tercer elemento ahora es 3.8
```

Las listas, a diferencia de las tuplas, permiten agregar nuevos valores:

```
mi_lista.append('Nuevo Dato')
```

Diccionarios

Mientras que a las listas y tuplas se accede solo y únicamente por un número de índice, los diccionarios permiten utilizar una clave para declarar y acceder a un valor:

```
mi_diccionario = {'clave_1': valor_1, 'clave_2': valor_2, \
                  'clave_7': valor_7}
print mi_diccionario['clave_2']  # Salida: valor_2
```

Un diccionario permite eliminar cualquier entrada:

```
del(mi_diccionario['clave_2'])
```

Al igual que las listas, el diccionario permite modificar los valores

```
mi_diccionario['clave_1'] = 'Nuevo Valor'
```


Comparte el conocimiento: Creative Commons Atribución-NoComercial-CompartirIgual 3.0

Estructuras de Control de Flujo

Una estructura de control, es un bloque de código que permite agrupar instrucciones de manera controlada. En este capítulo, hablaremos sobre dos estructuras de control:

- Estructuras de control condicionales
- Estructuras de control iterativas

Identación

Para hablar de estructuras de control de flujo en Python, es imprescindible primero, hablar de identación.

¿Qué es la identación? En un lenguaje informático, la identación es lo que la sangría al lenguaje humano escrito (a nivel formal). Así como para el lenguaje formal, cuando uno redacta una carta, debe respetar ciertas sangrías, los lenguajes informáticos, requieren una identación.

No todos los lenguajes de programación, necesitan de una identación, aunque sí, se estila implementarla, a fin de otorgar mayor legibilidad al código fuente. Pero **en el caso de Python, la identación es obligatoria**, ya que de ella, dependerá su estructura.

> **PEP 8: identación**
> Una identación de **4 (cuatro) espacios en blanco**, indicará que las instrucciones identadas, forman parte de una misma estructura de control.

Una estructura de control, entonces, se define de la siguiente forma:

```
inicio de la estructura de control:
    expresiones
```


Comparte el conocimiento: Creative Commons Atribución-NoComercial-CompartirIgual 3.0

Encoding

El **encoding** (o codificación) es otro de los elementos del lenguaje que no puede omitirse a la hora de hablar de estructuras de control.

> El **encoding** no es más que una **directiva que se coloca al inicio de un archivo Python, a fin de indicar al sistema, la codificación de caracteres utilizada en el archivo.**

```
# -*- coding: utf-8 -*-
```

utf-8 podría ser cualquier codificación de caracteres. Si no se indica una codificación de caracteres, Python podría producir un error si encontrara caracteres “extraños”:

```
print "En el Ñágara encontré un Ñandú"
```

Producirá un error de sintaxis: `SyntaxError: Non-ASCII character[...]`

En cambio, indicando el encoding correspondiente, el archivo se ejecutará con éxito:

```
# -*- coding: utf-8 -*-

print "En el Ñágara encontré un Ñandú"
```

Produciendo la siguiente salida:

```
En el Ñágara encontré un Ñandú
```

Asignación múltiple

Otra de las ventajas que Python nos provee, es la de poder asignar en una sola instrucción, múltiples variables:

```
a, b, c = 'string', 15, True
```

En una sola instrucción, estamos declarando tres variables: a, b y c y asignándoles un valor concreto a cada una:

```
>>> print a
string
>>> print b
15
>>> print c
True
```

La asignación múltiple de variables, también puede darse utilizando como valores, el

contenido de una tupla:

```
>>> mi_tupla = ('hola mundo', 2011)
>>> texto, anio = mi_tupla
>>> print texto
hola mundo
>>> print anio
2011
```

O también, de una lista:

```
>>> mi_lista = ['Argentina', 'Buenos Aires']
>>> pais, provincia = mi_lista
>>> print pais
Argentina
>>> print provincia
Buenos Aires
```


Comparte el conocimiento: Creative Commons Atribución-NoComercial-CompartirIgual 3.0

Estructuras de control de flujo condicionales

> *"[...] Los condicionales nos permiten comprobar condiciones y hacer que nuestro programa se comporte de una forma u otra, que ejecute un fragmento de código u otro, dependiendo de esta condición [...]"*
>
> *Cita textual del libro "Python para Todos" de Raúl González Duque (http://mundogeek.net/tutorial-python/)*

Las estructuras de control condicionales, son aquellas que nos permiten evaluar si una o más condiciones se cumplen, para decir qué acción vamos a ejecutar. **La evaluación de condiciones**, solo **puede arrojar** 1 de 2 resultados: **verdadero o falso** (True o False).

En la vida diaria, actuamos de acuerdo a la evaluación de condiciones, de manera mucho más frecuente de lo que en realidad creemos: ***Si** el semáforo está en verde, cruzar la calle. **Sino**, esperar a que el semáforo se ponga en verde.* A veces, también evaluamos más de una condición para ejecutar una determinada acción: *Si llega la factura de la luz **y** tengo dinero, pagar la boleta.*

Para describir la evaluación a realizar sobre una condición, se utilizan **operadores relacionales** (o de comparación):

OPERADORES RELACIONALES (DE COMPARACIÓN)

Símbolo	Significado	Ejemplo	Resultado
==	Igual que	5 == 7	Falso
!=	Distinto que	rojo != verde	Verdadero
<	Menor que	8 < 12	Verdadero
>	Mayor que	12 > 7	Falso
<=	Menor o igual que	12 <= 12	Verdadero
>=	Mayor o igual que	4 >= 5	Falso

Y para evaluar más de una condición simultáneamente, se utilizan **operadores lógicos**:

OPERADORES LÓGICOS

Operador	Ejemplo	Resultado*	
and (y)	5 == 7 and 7 < 12	0 y 0	Falso
	9 < 12 and 12 > 7	1 y 1	Verdadero
	9 < 12 and 12 > 15	1 y 0	Falso
or (o)	12 == 12 or 15 < 7	1 o 0	Verdadero
	7 > 5 or 9 < 12	1 o 1	Verdadero
xor	4 == 4 xor 9 > 3	1 o 1	Falso
(o excluyente)	4 == 4 xor 9 < 3	1 o 0	Verdadero

(*) 1 indica resultado verdadero de la condición, mientras que 0, indica falso.

Comparte el conocimiento: Creative Commons Atribución-NoComercial-CompartirIgual 3.0

Las estructuras de control de flujo condicionales, se definen mediante el uso de tres palabras claves reservadas, del lenguaje: **`if`** (si), **`elif`** (sino, si) y **`else`** (sino).

Veamos algunos ejemplos:

> Si semáforo esta en verde, cruzar la calle. Sino, esperar.

```
if semaforo == verde:
    print "Cruzar la calle"
else:
    print "Esperar"
```

> Si gasto hasta $100, pago con dinero en efectivo. Sino, si gasto más de $100 pero menos de $300, pago con tarjeta de débito. Sino, pago con tarjeta de crédito.

```
if compra <= 100:
    print "Pago en efectivo"
elif compra > 100 and compra < 300:
    print "Pago con tarjeta de débito"
else:
    print "Pago con tarjeta de crédito"
```

> Si la compra es mayor a $100, obtengo un descuento del 10%

```
importe_a_pagar = total_compra

if total_compra > 100:
    tasa_descuento = 10
    importe_descuento = total_compra * tasa_descuento / 100
    importe_a_pagar = total_compra – importe_descuento
```

Estructuras de control iterativas

A diferencia de las estructuras de control condicionales, las iterativas (también llamadas cíclicas o bucles), nos permiten ejecutar un mismo código, de manera repetida, mientras se cumpla una condición.

En Python se dispone de dos estructuras cíclicas:

- El bucle **while**
- El bucle **for**

Las veremos en detalle a continuación.

Bucle while

Este bucle, se encarga de ejecutar una misma acción "mientras que" una determinada condición se cumpla:

> Mientras que año sea menor o igual a 2012, imprimir la frase "Informes del Año *año*"

```
# -*- coding: utf-8 -*-

anio = 2001
while anio <= 2012:
    print "Informes del Año", str(anio)
    anio += 1
```

La iteración anterior, generará la siguiente salida:

```
Informes del año 2001
Informes del año 2002
Informes del año 2003
Informes del año 2004
Informes del año 2005
Informes del año 2006
Informes del año 2007
Informes del año 2008
Informes del año 2009
Informes del año 2010
Informes del año 2011
Informes del año 2012
```

Si miras la última línea:

```
anio += 1
```

Podrás notar que en cada iteración, incrementamos el valor de la variable que condiciona el bucle (`anio`). Si no lo hiciéramos, esta variable siempre sería igual a `2001` y el bucle se ejecutaría de forma infinita, ya que la condición (`anio <= 2012`) siempre se estaría cumpliendo.

Comparte el conocimiento: Creative Commons Atribución-NoComercial-CompartirIgual 3.0

Pero ¿Qué sucede si el valor que condiciona la iteración no es numérico y no puede incrementarse? En ese caso, podremos utilizar una estructura de control condicional, anidada dentro del bucle, y frenar la ejecución cuando el condicional deje de cumplirse, con la palabra clave reservada break:

```
while True:
    nombre = raw_input("Indique su nombre: ")
    if nombre:
        break
```

El bucle anterior, incluye un condicional anidado que verifica si la variable nombre es verdadera (solo será verdadera si el usuario tipea un texto en pantalla cuando el nombre le es solicitado). Si es verdadera, el bucle para (break). Sino, seguirá ejecutándose hasta que el usuario, ingrese un texto en pantalla.

Bucle for

El bucle for, en Python, es aquel que nos permitirá iterar sobre una variable compleja, del tipo lista o tupla:

> Por cada nombre en mi_lista, imprimir nombre

```
mi_lista = ['Juan', 'Antonio', 'Pedro', 'Herminio']
for nombre in mi_lista:
    print nombre
```

> Por cada color en mi_tupla, imprimir color

```
mi_tupla = ('rosa', 'verde', 'celeste', 'amarillo')
for color in mi_tupla:
    print color
```

En los ejemplos anteriores, nombre y color, son dos variables declaradas en tiempo de ejecución (es decir, se declaran dinámicamente durante el bucle), asumiendo como valor, el de cada elemento de la lista (o tupla) en cada iteración.

Otra forma de iterar con el bucle for, puede emular a while:

> Por cada año en el rango 2001 a 2013, imprimir la frase "Informes del Año *año*"

```
# -*- coding: utf-8 -*-
for anio in range(2001, 2013):
    print "Informes del Año", str(anio)
```