Fundamentos de Programación con Python

Bloque 1:

Introducción a la programación con Python

Tabla de contenido

Un poco sobre Python

- Este lenguaje fue creado a principios de los noventa por Guido van Rossum en los Países Bajos.
- **Python** es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.
- Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.
- Es administrado por la Python Software Foundation. Posee una licencia de código abierto

En este manual se irán exponiéndolos fundamentos básicos de la programación, los elementos y conceptos que necesitarás aprender para poder desarrollar programas básicos en Python.

Variables

Las **variables** serán datos que almacenaremos en la memoria para operar con ellos, por tanto, podrán variar a lo largo de la ejecución del programa.

Para declarar una variable será necesario especificar un nombre (conocido como **identificador**
y que decidirá el programador) y el tipo de dato que va a guardar.

Ejemplos de variables en Python:

```
>>> c = "Hola Mundo" # cadenas de caracteres
>>> type(c) # comprobar tipo de dato
<type 'str'>
>>> e = 23 # número entero
>>> type(e) # comprobar tipo de dato
<type 'int'>
```

En Python, a diferencia de muchos otros lenguajes, no se declara el tipo de la variable al crearla. En *Java*, por ejemplo, definir una variable seria así:

```
String c = "Hola Mundo";
int e = 23;
```

También nos ha servido el pequeño ejemplo para presentar los comentarios en línea en Python: cadenas de caracteres que comienzan con el carácter `#` y que Python ignora totalmente.

Tipos de datos

Los **tipos de datos** son el conjunto específico de valores de los datos y el conjunto de operaciones que actúan sobre esos datos. Podemos distinguir:

- Tipos de datos básicos o primitivos:
  - Numéricos: enteros, reales, complejos.
  - Texto: carácter, cadenas de caracteres (texto)
  - Lógicos: booleanos (True/False)
- Tipos de datos complejos o estructurados: listas, tuplas y diccionarios.

Cuando se intenta hacer una operación con una variable de un tipo de dato que no está permitida nos dará un error al compilar el programa. Por ejemplo, no se podrá sumar un número con una cadena.

Los lenguajes de programación cuentan con funciones internas para conocer el tipo de dato que contiene una variable o poder convertir una variable de un tipo a otro. Cuando el programador, a través de estas funciones, cambia el tipo de la variable se está realizando una conversión explícita denominada casting.

Identificadores

Como ya se ha indicado anteriormente un **identificador** será el nombre que el programador asigne a una variable, constante, función, etc.

Cada lenguaje de programación tiene sus restricciones en cuanto al conjunto de letras, números y caracteres especiales que se pueden emplear en el identificador.

Así mismo, hay lenguajes de programación que distinguen entre mayúsculas y minúsculas. Por tanto, el identificador Numero y numero pueden ser diferentes.

De la misma manera, cada lenguaje de programación reserva un conjunto de palabras reservadas que tienen un significado propio para el lenguaje y no se podrá utilizar como identificador

NORMAS IDENTIFICADORES EN PYTHON

- Un identificador comienza con una letra o con guión bajo (_) y luego sigue con una secuencia de letras, números y guiones bajos.
- Los espacios no están permitidos dentro de los identificadores. Tampoco se permite utilizar los signos de puntuación y caracteres: @, $ y %
- OJO Python es case sensitive. No es lo mismo el identificador edad y Edad

EJEMPLOS IDENTIFICADORES CORRECTOS EN PYTHON

```
edad
edad12t
edad
```

EJEMPLOS DE IDENTIFICADORES INCORRECTOS EN PYTHON

```
edad compra12t
8edad
edad\%
```

NOTA: Por convención, no usaremos identificadores que empiezan con mayúscula. Es una buena práctica aplicar las convenciones establecidas.

PALABRAS RESERVADAS EN PYTHON

Todos los lenguajes de programación tienen sus propias palabras, las cuales no podemos utilizar como nombres variables, funciones o clases. Python reserva 31 palabras para describir la estructura del programa, y no permite que se usen como identificadores. Las palabras reservadas son:

False	*class*	*from*	*or*
None	*continue*	*global*	*pass*
True	*def*	*if*	*raise*

and	*del*	*import*	*return*
as	*elif*	*in*	*try*
assert	*else*	*is*	*while*
async	*except*	*lambda*	*with*
await	*finally*	*nonlocal*	*yield*
break	*for*	*not*	

Disponemos de una función en Python que nos muestra las palabras que tiene reservadas

```
help("keywords")
```

Reglas en Python

Algunas reglas y **convenciones para los identificadores** de las variables y constantes:

- Nunca usar símbolos especiales como !, @, #, $, %, etc.
- El primer carácter no puede ser un número o dígito.
- Las constantes son colocadas dentro de módulos Python y significa que no puede ser cambiado.
- Los nombres de constante y variable debería tener la combinación de letras en minúsculas (de *a* a la *z*) o MAYÚSCULAS (de la *A* a la *Z*) o dígitos (del *0* al *9*) o un `underscore` (_). Por ejemplo:
    - snake_case
    - MACRO_CASE
    - camelCase
    - CapWords
- Los nombres que comienzan con guión bajo (simple _ o doble) se reservan para variables con significado especial
- No pueden usarse como identificadores, las palabras reservadas.

Alcance de variables

Las variables en Python son locales por defecto. Esto quiere decir que las variables definidas y utilizadas en el bloque de código de una función, sólo tienen existencia dentro de la misma, y no interfieren con otras variables del resto del código. Más adelante, se explicarán qué son las funciones.

En caso de que sea conveniente o necesario, una variable local puede convertirse en una variable global declarándola explícitamente como tal con la sentencia global.

Constantes

Una constante es un tipo de variable la cual no puede ser cambiada. Eso es muy de ayuda pensar las constantes como contenedores que contienen información el cual no puede ser cambiado después.

La constante vendrá definida por un nombre (denominado identificador) y un valor. Por ejemplo:

```
PI = 3.14159
```

Las constantes no tienen que ser valores numéricos, pueden ser también cadenas. Por ejemplo:

```
Warning = ”Prohibido continuar. Ha sucedido un error”
```

Generalmente, dependiendo de las reglas de estilo de cada lenguaje de programación el nombre de las constantes es en mayúsculas

En Python, las constantes son usualmente declaradas y asignadas en un módulo. Aquí, el módulo significa un nuevo archivo que contiene variables, funciones, etc.; el cual es importado en el archivo principal. Dentro del módulo, las constantes son escritas en letras MAYÚSCULAS

```
#Fichero Constantes.py
IP_DB_SERVER = "127.0.0.1"
PORT_DB_SERVER = 3307
USER_DB_SERVER = "root"
PASSWORD_DB_SERVER = "123456"
DB_NAME = "nomina"
```

```
#Fichero main.py

import constantes #para poder referenciar los elementos del otro fichero

print(IP_DB_SERVER)
print(USER_DB_SERVER)
```

Entrada/salida básica

Para comenzar a realizar los primeros ejemplos en Python se empleará una función propia del lenguaje para poder solicitar información al usuario. Cada lenguaje de programación tiene sus propias funciones para realizar esta tarea. Veamos algunos ejemplos en Python:

```
nombre = input(“Escribe tu nombre, por favor: “)
```

De la misma manera para mostrar un mensaje por pantalla podremos emplear la función print

```
print(“Bienvenido al curso”)
```

La sintaxis de print, la encontrarás de diferentes maneras. No tan sencilla como el ejemplo anterior.

Tipos de datos simples y estructurados

Tipo número

Estos tipos de datos se crean mediante literales numéricos y se devuelven como resultados por operadores aritméticos y funciones aritméticas integradas. Los objetos numéricos son inmutables; Una vez creado su valor nunca cambia.

ENTEROS

Los números enteros son aquellos que no tienen decimales, tanto positivos como negativos (además del cero). En Python se pueden representar mediante el tipo `int` (de integer, entero)

Nota: En Python 3 deja de existir el tipo entero largo

(long) Ejemplos:

```
entero = 7
print(entero, type(entero))
```

COMA FLOTANTE

Los números reales son los que tienen decimales. En Python se expresan mediante el tipo
`float`.

Ejemplo

s:

```
real=0.56
print(real, type(real))
real=0.1e-3
print(real, type(real))
```

COMPLEJOS

Los números complejos son aquellos que tienen parte imaginaria. Si no conocías de su existencia, es más que probable que nunca lo vayas a necesitar,

Ejemplo:

```
complejo = 3 + 7.8j
print(complejo, type(complejo))
```

CONVERTIR DE TIPO

Para convertir a tipos numéricos debe usar las siguientes funciones integradas en el intérprete Python:

- La función int() devuelve un tipo de datos número entero.
- La función float() devuelve un tipo de datos número entero float.
- La función complex() devuelve un tipo de datos número complejo.

Tipos booleanos

El tipo booleano sólo puede tener dos valores: `True` (verdadero) y `False` (falso). Estos valores son especialmente importantes para las expresiones condicionales y los bucles, como verá más adelante.

Ejemplo:

```
bool = True
print(bool, type(bool)
```

Tipo cadena de caracteres

Las cadenas de caracteres son secuencias que contienen caracteres encerrado entre

comillas. Ejemplo:

```
cadena=”Hola”
print(cadena, type(cadena))
```

Se pueden hacer operaciones con cadenas. Con el operador + se concatenan cadenas y con el operador * se repite la cadena

Python viene con cuatro estructuras de datos incorporadas que puedes usar para mantener cualquier colección de objetos. Son: lista, tupla, diccionario, y conjunto, los cuales se explican a continuación

Tipos datos complejos

Dispones de tipos de datos en Python que nos permitirá estructurar la información.

Para entender mejor estas estructuras es muy conveniente probarlas en http://pythontutor.com/ una página Web donde podemos ejecutar código Python y nos visualizará cómo se estructuran estos tipos.

Nota: si tienes grandes dificultades al comienzo, esta página es muy recomendable porque puedes probar cualquier código Python para comprender cómo se ejecuta el código.

Tipo listas

Las listas en Python son:

- **heterogéneas**: pueden estar conformadas por elementos de distintos tipos, incluidos otras listas.
- **mutables**: sus elementos pueden modificarse.

Una lista en Python es una estructura de datos formada por una secuencia ordenada de objetos.

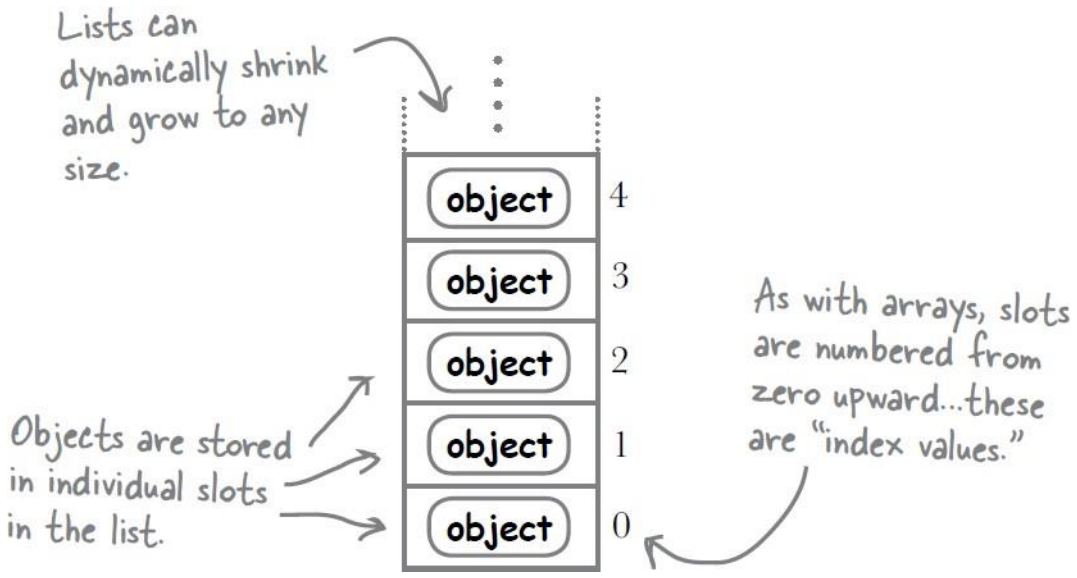


Ilustración 1 Listas Python (imagen obtenida del libro Head First Python, de Paul Barry Ed. O'Reilly)

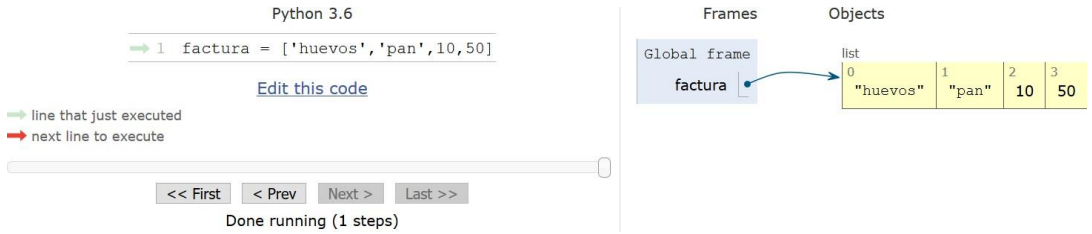
Los elementos de la lista están entre [] separados por comas.

Los elementos de una lista pueden accederse mediante su índice, siendo 0 el índice del primer elemento.

Ejemplo de creación de una lista con valores:

```
factura = ['huevos','pan',10,50] #creación de una lista con valores
```

Probamos la anterior línea en http://pythontutor.com/ para que comprendas visualmente cómo es una lista Python



Ejemplo de como mostrar los elementos de una lista:

```
print(factura)
print(factura[0])
print(factura[1])
print(factura[2])
print(factura[3])
```

Ejemplo de creación de una lista vacía:

```
precios=[] #creación de una lista vacía
```

MÉTODOS DE LISTAS

- append(). Este método agrega un elemento al final de una lista.
- count(). Este método recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la lista.
- extend(). Este método extiende una lista agregando un iterable al final.
- index(). Este método recibe un elemento como argumento, y devuelve el índice de su primera aparición en la lista.
- insert(). Este método inserta el elemento x en la lista, en el índice i.
- pop(). Este método devuelve el último elemento de la lista, y lo borra de la misma.
- remove(). Este método recibe como argumento un elemento, y borra su primera aparición en la lista.
- reverse(). Este método invierte el orden de los elementos de una lista.
- sort(). Este método ordena los elementos de una lista.
- list(). Para convertir un tipo a tipo lista

Tipo tuplas

Las tuplas es una colección de objetos ordenada e inmutable.

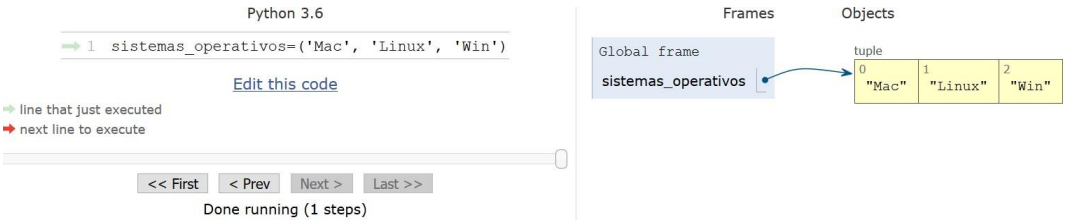
Las tuplas son objetos de tipo *secuencia*, específicamente es un tipo de dato lista inmutable. Esta no puede modificarse de ningún modo después de su creación.

SE utilizan los paréntesis (y sus elementos separados por comas.

Ejemplo

```
sistemas_operativos=('Mac', 'Linux', 'Win')
print(sistemas operativos, type(sistemas operativos))
```

Visualización en Python Tutor:



Son muy similares a las listas y comparten varias de sus funciones y métodos integrados, aunque **su principal diferencia es que son inmutables**. El objeto de tipo *tupla* integra una serie de métodos integrados a continuación

MÉTODOS DE TUPLAS

- count() Este método recibe un elemento como argumento, y cuenta la cantidad de veces que aparece en la tupla.
- index(). Comparte el mismo método index() del tipo lista. Este método recibe un elemento como argumento, y devuelve el índice de su primera aparición en la tupla.
- tuple() para convertir a tipo tupla

Tipo diccionarios

La principal característica de lso diccionarios es que los datos se almacenan asociados a una clave de tal forma que se crea una asociación del tipo *clave:valor* para cada elemento almacenado. Es decir, en lugar de acceder a la información mediante el índice numérico, como es el caso de las listas y tuplas, es posible acceder a los valores a través de sus claves, que pueden ser de diversos tipos.

Los elementos almacenados no están ordenados. El orden es indiferente a la hora de almacenar la información en un diccionario.

Las claves son únicas dentro de un diccionario, es decir que no puede haber un diccionario que tenga dos veces la misma clave, si se asigna un valor a una clave ya existente, se reemplaza el valor anterior.

Los diccionarios son:

- heterogéneos. Los datos que pueden almacenar son de distinto tipo
- mutables
- se utilizan con { } separados los *clave-valor* por comas

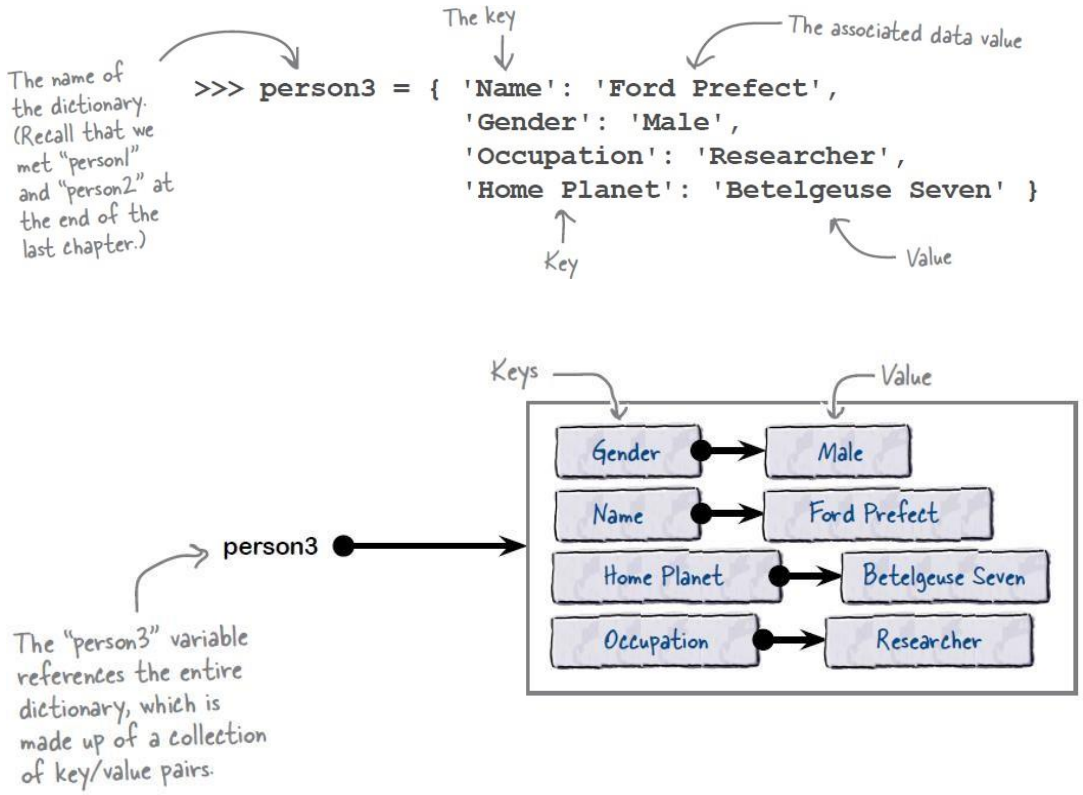


Ilustración 2 Diccionario en Python (imagen obtenida del libro Head First Python, de Paul Barry Ed. O'Reilly)

Ejemplo:

```
datos_basicos = {
    "nombres":"Leonardo Jose",
    "apellidos":"Caballero Garcia",
    "clave":"26938401",
    "fecha_nacimiento":"03/12/1980",
    "lugar_nacimiento":"Maracaibo, Zulia, Venezuela",
    "nacionalidad":"Venezolana",
    "estado_civil":"Soltero"
}
print (datos_basicos, type(datos_basicos))

print ("\nDetalle del diccionario")
print ("========================")
print ("\nClaves de diccionario:", datos_basicos.keys())
print ("\nValores de diccionario:", datos_basicos.values())


print ("\nDatos de participante")
print ("---------------------------")
print ("Clave de identidad: ", datos_basicos['clave'])
print ("Nombre completo: " + datos_basicos['nombres'] + " ")
```

```
    print ("Nacionalidad:", datos_basicos['nacionalidad'])
    print ("Estado civil:", datos_basicos['estado_civil'])
```

La estructura en Python Tutor es:

No hay una forma directa de acceder a una clave a través de su valor, y nada impide que un mismo valor se encuentre asignado a distintas claves

La información almacenada en los diccionarios no tiene un orden particular. Ni por clave ni por valor, ni tampoco por el orden en que han sido agregados al diccionario.

Cualquier variable de tipo inmutable, puede ser clave de un diccionario: cadenas, enteros, tuplas (con valores inmutables en sus miembros), etc. No hay restricciones para los valores que el diccionario puede contener, cualquier tipo puede ser el valor: listas, cadenas, tuplas, otros diccionarios, objetos, etc.

MÉTODOS DE DICCIONARIOS

- keys() Devolver todas las claves del diccionario
- values() devolverá todos los valores del diccionario
- del doccinario[clave] para eliminar ese dato del diccionario
- in para determinar si un valor está dentro
- clear() elimina todas las claves y valores
- get(clave [,"mensaje sino existe la clave"] obtener el valor de una clave

Operadores

Aritméticos

Los operadores aritméticos en Python son:

Operador	Descripción	Ejemplo
+	Suma	`>>> 3 + 2` `5`
-	Resta	`>>> 4 - 7` `-3`
-	Negación	`>>> -7`

		`-7`
*	Multiplicación	`>>> 2 * 6` `12`
**	Exponente	`>>> 2 ** 6` `64`
/	División	`>>> 3.5 / 2` `1.75`
//	División entera	`>>> 3.5 // 2` `1.0`
%	Módulo	`>>> 7 % 2` `1`

Relacionales o comparación

Los operadores de comparación en Python son:

Operador	**Descripción**	**Ejemplo**
==	¿son iguales a y b?	`>>> 5 == 3` `False`
!=	¿son distintos a y b?	`>>> 5 != 3` `True`
<	¿es a menor que b?	`>>> 5 < 3` `False`
>	¿es a mayor que b?	`>>> 5 > 3` `True`
<=	¿es a menor o igual que b?	`>>> 5 <= 5` `True`
>=	¿es a mayor o igual que b?	`>>> 5 >= 3` `True`

Lógicos

Los operadores lógicos permiten encadenar varias condiciones.

Operador	Descripción	Ejemplo
and	¿se cumple a y b?	`>>> True and False` `False`
or	¿se cumple a o b?	`>>> True or False` `True`
not	No al valor	`>>> not True` `False`

Asignación

Operador	Descripción	Ejemplo
=	asigna valor a una variable	`>>> r = 5` `>>> r1 = r`

+=	suma el valor a la variable	`>>> r = 5`
		`>>> r += 10; r`
		`15`

-=	resta el valor a la variable	`>>> r = 5`
		`>>> r -= 10; r`
		`-5`

*=	multiplica el valor a la variable	`>>> r = 5`
		`>>> r *= 10; r`
		`50`

/=	divide el valor a la variable	`>>> r = 5`
		`>>> r /= 10; r`
		`0`

**=	calcula el exponente del valor de la variable	`>>> r = 5`
		`>>> r **= 10; r`
		`9765625`

//=	calcula la división entera del valor de la variable	`>>> r = 5`
		`>>> r //= 10; r`
		`0`

%=	devuelve el resto de la división del valor de la variable	>>> r = 5
		>>> r %= 10; r
		5

Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales.

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas. Una expresión consta de operandos y operadores. Según sea el tipo de objetos que manipulan, las expresiones se clasifican en:

- aritméticas,
- relacionales,
- lógicas,
- carácter.

El resultado de la expresión aritmética es de tipo numérico; el resultado de la expresión relacional y de una expresión lógica es de tipo lógico; el resultado de una expresión carácter es de tipo carácter.

Es importante conocer el orden de precedencia de las expresiones para poder evaluarla correctamente.

Comentarios

Es importante que los programadores documenten los programas:

- Identificar el autor, fecha, modificaciones, objetivo
- Clarificar el código, ayudará a mantenerlo dentro de un gran equipo de programadores
- Activar o desactivar temporalmente líneas de código para depurarlo

Cada lenguaje de programación tiene sus propios caracteres para comentar. En el caso de Python:

- Solo existen los comentarios para una línea con el carácter #
- Ojo las docstrings (“ “ “) no es forma de comentar los programas. Esto se utilizará el inicio de cada pieza de programación y se mostrará en la documentación al utilizar el comando help. Ejemplo de una función en Python que se ha utilizado docstrings

```
def es_anterior (fecha1 , fecha2 ):
""" Fecha , Fecha -> bool
OBJ : Calcula si fecha1 es anterior a fecha2 """
if ( fecha2 . anno > fecha1 . anno ):
return True
elif ( fecha1 . anno > fecha2 . anno ):
return False
else : # si los dos annos son iguales
if ( fecha2 . mes > fecha1 . mes):
...
etc.
```

Estructuras de control selectivas e iterativas

Un programa durante su ejecución va pasando por los diferentes pasos que ha especificado el programador. Estos pasos pueden ser:

- Secuenciales: se ejecutará un paso o línea de programación, luego la siguiente y así sucesivamente.
- Selectivos: llegamos a un paso o línea de programación que contiene una condición o expresión lógica que en función de si es verdadera o falsa puede que se ejecuten unos pasos u otros.
- Iterativos: los pasos a ejecutar se repetirán

Este tipo de pasos son las estructuras de control de programación es lo que definirá el flujo de control del programa: la secuencia de ejecución de las instrucciones del programa.

Estructuras de control selectivas

Una estructura selectiva permite, de acuerdo a una condición, ejecutar o no ciertas instrucciones.

Como el objetivo es aprender a programar en Python, veamos como pueden ser las estructuras selectivas en este lenguaje.

if

Condición es de tipo booleano, y cuando ésta se cumple se ejecuta el bloque.

```
if condicion:
    bloque
```

Nota: Cada vez que una sentencia acaba con dos puntos Python espera que la sentencia o
sentencias que le siguen aparezcan con un mayor sangrado. Es la forma de marcar el inicio y el fin de una serie de sentencias que dependen de otra.

Excepcion: si solo hay una sentencia que depende de otra, pueden escribirse ambas en la misma línea.

Ejemplo: Pedir un número al usuario y si es negativo mostrar su valor absoluto.

if else

Condición es de tipo booleano, cuando ésta se cumple se ejecuta el bloque 1, cuando no se cumple se ejecuta el bloque 2.

```
if condicion:
    bloque 1
else:
    bloque 2
```

Además, tenemos la siguiente estructura, que abrevia el if else if:

Ejemplo: Pedir un número al usuario e indicar con un mensaje si es par o impar.

if elif [else]

```
if condicion 1:
    bloque 1
elif condicion 2:
    bloque 2
```

```
elif condicion 3:
    bloque 3
else:
    bloque 4
```

Ejemplo: Pedir dos números al usuario e indicar qué número es mayor que otro o si son iguales

Estructuras de control iterativas

Una estructura iterativa (bucle) engloba un conjunto de instrucciones que se ejecutan ninguna, una o tantas veces como indique una determinada condición.

Conceptualmente existen 3 tipos de bucles:

- Desde (número de iteraciones conocido)
- Mientras (0 o más iteraciones) y
- Repetir (1 o más iteraciones)

Bucle for

- Permite ejecutar una sentencia o bloque de sentencias un numero conocido de veces.
- Itera sobre una lista de valores conocidos, bien numéricos (bastante frecuente) o de otro tipo.
- Una variable de control toma sucesivamente todos los valores de la lista.

El cuerpo del bucle se ejecuta tantas veces como elementos tenga el elemento recorrible (elementos de una lista o de un range(), caracteres de una cadena, etc.).

Veamos en Python como es la sintaxis de un *bucle for*:

for

```
for variable in elemento iterable (lista, cadena, range, etc.):
    cuerpo del bucle
```

Ejemplo: Mostrar por pantalla tres veces el mensaje Hola

Range

En la estructura de los *bucles for* a menudo se emplea una función Python que se llama
range. Hacemos un inciso en este punto para explicar en qué

consiste. La función *range* se emplea para generar una lista de

enteros

```
range(10)
```

Para comprenderlo, lo mejor es probarlo:

```
range (5) : [ 0, 1, 2, 3, 4 ] desde 0 hasta el numero menos 1
range (2,5) : [ 2, 3, 4 ] de 2 a 5 - 1
range (3, 10, 2) : [ 3, 5, 7, 9 ] de 3 a 10 menos 1 de 2 en 2
```

¿Qué obtendríamos de los siguientes ejemplos?

```
for i in range (10):
    print (i)
```

```
for i in range (1 ,20):
     print (i, end ="")
```

```
words = ['cat ', 'dog ', 'lion ']
for w in words :
    print (w, end =''/'')
```

Nota: cuando usamos la función print con el argumento end le indicamos como deseamos
finalizar la línea, sino se especifica este argumento por defecto es un salto de línea “\n”

```
x = 0
for i in range (1, 20, 2):
    x += i
print (x)
```

Ejercicio 1: pedir un número al usuario y mostrar la tabla su tabla de multiplicar.

Ejercicio 2: mostrar las tablas de multiplicar de los números pares

Bucle while

El bucle while se ejecuta mientras la condición sea cierta, si la condición es falsa al inicio, el bloque no se ejecuta y se pasan a ejecutar las sentencias que le siguen.

En Python, cualquier valor entero distinto de cero es verdadero y 0 es falso. La condición también puede ser una lista o cualquier secuencia, siendo la secuencia vacía falsa.

El cuerpo del bucle debe estar sangrado, ya que de este modo Python agrupa las

sentencias. En Python la sintaxis de un bucle while es:

```
while condicion:
    bloque
```

ADVERTENCIA: habrá que modificar en algún momento la condición para que el bucle deje de ejecutarse, sino será un bucle infinito.

Ejemplo: Cuenta atrás del 10 al 1 para el despegue:

```
n = 10
while n > 0:
   print n
   n = n -1
print ('Despegue!')
```

Ejercicio: pedir números al usuario hasta que escriba un número par

Funciones

La mayoría de las sentencias en un típico programa en Pyhton están agrupadas y organizadas en funciones.

Una función es un grupo de sentencias que se ejecutan cuando se las invoca. Python proporciona muchas funciones integradas y permite a los programadores definir sus propias funciones.

Cuando llamamos o invocamos a la función podemos enviar argumentos sobre los cuales la función puede trabajar.

En Python ujna función siempre devuelve un valor de resultado, o bien None o bien un dato.

En Python las funciones son tratadas como objetos, lo que supone que se puede enviar como parámetro de una función otra función.

DEFINICIÓN DE FUNCIONES

```
def nombre-funcion (parámetros):
      instrucciones
```

Nombre-funcion: es el identificador de la función.

Parámetros es una lista opcional de identificadores que podemos enviar a la función. Si la función tiene más de un parámetro, estos se separan por comas

Por último, tenemos el cuerpo de la función, con las instrucciones necesarias para realizar su tarea. Al final de la función se devolverá el valor con la palabra return

Ejemplo de función:

```
def double (x):
      return x*2
```

PARÁMETROS

- Paso por valor: se envía simplemente el valor de la variable, en cuyo caso el módulo no puede modificar la variable, pues el módulo solo conoce su valor, pero no la variable que lo almacenaba.
- Paso por referencia: se envía la dirección de memoria de la variable, en cuyo caso el módulo s que puede modificar la variable.

En Python no se hace ni una cosa ni otra. En Python cuando se envía una variable como argumento en una llamada a un módulo lo que se envía es la referencia al objeto al que hace referencia la variable. Dependiendo de si el objeto es mutable o inmutable, el módulo podrá modificar o no el objeto.

```
def aumenta(x):
    print(id(x))
    x += 1
    print(id(x))
    return x

a = 3
print(id(3), id(4))
print(id(a))
print(aumenta(a))
print(a)
print(id(a))
```

```
505894336 505894352
505894336
505894336
505894352
4
3
505894336
```

En el ejemplo siguiente, la variable enviada a la funcin es una variable que hace referencia a un objeto mutable

```
def aumenta(x):
    print(id(x))
    x += [1]
    print(id(x))
    return x

a = [3]
print(id(a))
print(aumenta(a))
print(a)
print(id(a))
```

```
45688512
45688512
45688512
[3, 1]
[3, 1]
45688512
```

Funciones predefinidas

Todos los lenguajes de programación disponen de un conjunto de funciones ya creadas que nos facilitan realizar ciertas operaciones. En este apartado, se mostrarán algunas funciones que nos serán de utilidad a lo largo del curso.

Funciones de cadenas en Python

Función	Utilidad	Ejemplo	Resultado
print()	Imprime en pantalla el argumento.	**print («Hola»)**	«Hola»
len()	Determina la longitud en caracteres de una cadena.	**len(«Hola Python»)**	11

Función	Utilidad	Ejemplo	Resultado
join()	Convierte en cadena utilizando una separación	**Lista = [‘Python’, ‘es’]** **‘-‘.join(Lista)**	‘Python-es’
split()	Convierte una cadena con un separador en una lista	**a = («hola esto sera una lista»)** **Lista2 = a.split()** **print (Lista2)**	[‘hola’, ‘esto’, ‘sera’, ‘una’, ‘lista’]
replace()	Reemplaza una cadena por otra	**texto = «Manuel es mi amigo»** **print (texto.replace (‘es’, ‘era’))**	Manuel era mi amigo
upper()	Convierte una cadena en Mayúsculas	**texto = «Manuel es mi amigo»** **texto.upper()**	‘MANUEL ES MI AMIGO’
lower()	Convierte una cadena en Minúsculas	**texto = «MaNueL eS mI AmIgO»** **texto.lower()**	‘manuel es mi amigo’

Funciones numéricas en Python

Función	Utilidad	Ejemplo	Resultado
range()	Crea un rango de números	**x = range (5)** **print (list(x))**	[0, 1, 2, 3, 4]
str()	Convierte un valor numérico a texto	**str(22)**	’22’
int()	Convierte a valor entero	**int(’22’)**	22
float()	Convierte un valor a decimal	**float(‘2.22’)**	2.22
max()	Determina el máximo entre un grupo de números	**x = [0, 1, 2]** **print (max(x))**	2
min()	Determina el mínimo entre un grupo de números	**x = [0, 1, 2]** **print (min(x))**	0
sum()	Suma el total de una lista de números	**x = [0, 1, 2]** **print (sum(x))**	3

Otras funciones útiles en Python

Función	Utilidad	Ejemplo	Resultado
list()	Crea una lista a partir de un elemento	**x = range (5)** **print (list(x))**	[0, 1, 2, 3, 4]
tuple()	Crea o convierte en una tupla	**print(tuple(x))**	(0, 1, 2, 3, 4)
open()	Abre, crea, edita un elemento (archivo)	**with open(«Ejercicios/Ejercicio.py», «w») as variables: variables.writelines(«Eje»)**	Crea el archivo «Ejercicio.py» con el contenido «Eje»
ord()	Devuelve el valor ASCII de una cadena o carácter.	**print(ord('A'))**	65
round()	Redondea después de la coma de un decimal	**print (round(12.723))**	13
type()	Devuelve el tipo de un elemento	**type(x)**	<class 'range'>
input()	Permite la entrada de datos al usuario en Python 3	**y = int(input(«Ingrese el número»))** **print (y)**	3 3

Para consultar todas las funciones predefinidas de Python consultar: https://docs.python.org/3/library/functions.html