Portada

Blas García

pythontutor.com

16/09/2016

1. Nociones qué es un PC
2. Nociones Binario (potencias de 2, dividir entre 2 para dec->bin….)

#### Tipos de Código

##### Lenguaje Bajo Nivel

*Código Máquina*  
 Programa escrito en código binario, capaz de ser reconocido directamente por el  
microprocesador.  
Es completamente dependiente de la máquina, lo que quiere decir que cada microprocesador tiene su juego de instrucciones/órdenes propia (las direcciones de memoria de las órdenes son distintas entre microprocesadores).

*Assembler / Lenguaje Ensamblador*  
 Sustituye las órdenes binarias del microprocesador por palabras mnemotécnicas (SUM, ACU, etc). Requiere de un programa traductor ensamblador para traducir a código máquina.

Al igual que el código máquina, es dependiente del microprocesador, el traductor ensamblador ha de estar adaptado al microprocesador concreto.

##### Lenguaje Alto Nivel

Lenguajes de programación tipo sintáctico, matemático, etc. Ejemplos de estos lenguajes son Python, Java, C/++, MatLab…

Al contrario del lenguaje a bajo nivel, el lenguaje a alto nivel son independientes a la máquina física y es muy cercano al usuario. Requieren de algún sistema de interpretación al código máquina, o bien dependiente al sistema operativo (Compilador: C/++) o bien a un emulador (Intérprete: Java).

Un compilador es un traductor clásico de lenguaje de alto nivel a código máquina. El compilador es dependiente al sistema operativo. El ejecutable generado es un código enlazado/linkado.  
[Traduce el código fuente a código máquina produciendo un ejecutable]

Los intérpretes viene a ser una máquina virtual que hace independiente las aplicaciones que se monten sobre dicho intérprete del sistema operativo que esté por debajo de él.  
[Traduce el código fuente a código intermedio |bytecode| línea a línea, ejecutándola “on the fly” por la máquina virtual]

El compilador, pese a tener una traducción más lenta, la ejecución es más rápida que la del intérprete, sin embargo el intérprete tiene la ventaja de tener un código fuente ejecutable en cualquier[[1]](#footnote-0) sistema operativo.

20/09/2016

#### Introducción Python

Se trata de un lenguaje interpretado, relativamente nuevo, y pese a su simpleza es muy potente.

Es un lenguaje de propósito general, es decir, no está diseñado para un tipo específico de problema, sino que es abierto y versátil.

Cuenta con una comunidad de desarrollo muy amplia y, aunque el propio lenguaje cuenta con copyright, su distribución es libre y abierta.

Tiene un desarrollo muy avanzado, por lo que cuenta con un amplio catálogo de librerías.

Es multiparadigma, que viene a decir que permite seguir una programación funcional estructurada de muy alto nivel o programación orientada a objetos.

Una funcionalidad es que tiene tipado automático, es decir, no es necesario declarar el tipo antes de usarlo, aún y así no hace traducción automática de tipos. Para traducir tipos hay que usar las funciones concretas de conversión de tipo.

Al igual que Java, tiene su propio Garbage Collector, que se encarga de ir limpiando la memoria.

22/09/2016

Una característica importante de Python es que los Strings son indexables (String = String[]).

*str = ‘Hola’ str[0] : ‘H’ str[1] : ‘o’ str[2] : ‘l’ str[3] : ‘a’*

*Si se especifica un índice fuera del rango salta error.*

Dichas Strings son INMUTABLES, es decir, no son reasignables, sólo es posible reasignar su variable.

*str = ‘Hola’ str[1] = ‘e’ # 'str' object does not support item assignment*

Todas las secuencias se numeran desde el CERO. Admite índices negativos, en cuyo caso -1 es el último carácter de la cadena.

*str = ‘Hola’ str[-1] : ‘a’ str[-2] : ‘l’ str[-3] : ‘o’ str[-4] : ‘H’*

Para sacar un segmento o substring se usa el slicing, que sigue el principio de Inclusión-Exclusión

*str = ‘Hola’ str[0:2] : ‘Ho’ # 0 incluído, 2 excluído*

*El slicing no produce error de rango: str[1:16] : ‘ola’  
El número más pequeño ha de ir siempre a la izquierda, en caso contrario se devuelve una cadena vacía.*

*Si no se especifica un límite inferior o superior se considera que se toma desde el principio o hasta el final.*

*str[:3] : ‘Hol’ str[2:] : ‘la’*

Una lista representa una variable estructurada indexable de N elementos, sin tamaño fijo. Lo que viene a funcionar como un Array+List de Java pero más chulo. Es mutable y multitipo.

*Lista = ['Hola', 'Pepe', 37, 5.4]*

Se puede usar el mismo algoritmo de los contadores y acumuladores a las cadenas para ir acumulando caracteres:

* c = c + 1 (c +=1 ) # Contador
* a = a + t (a += t) # Acumulador
* str = str + con (str += con) # Concatenación de strings

06/10/2016

Dentro de las funciones podemos incluir un docstring, que es un comentario que sirve para explicar qué es lo que hace a función (tipo Javadoc pero dentro de la función, no fuera). Además Python cuenta con un sistema de autodocumentación que estudia los docstrings de las funciones.

##### 

##### 

#### Problemas:

1- Tengo un folio que mide 0.1mm de espesor. ¿Cuál será su espesor tras la 50º doblez? Dar la respuesta en Kilómetros

>>> 0.1\*2\*\*50  
 112589990684262.4 #mm = 11258999.06842624km

2- Suponiendo que cadena vale “Hola Pepe” hay que asignar una nueva variable que sea “Pepe Hola”

>>> cadena = "Hola Pepe"

'Hola Pepe'

>>> cadena2 = cadena[:4] + ' ' + cadena[5:]

'Pepe Hola'

#### Ejercicios:

1. Escribir un programa que imprima lo números pares hasta el 200
2. Escribir un programa que pida un número al usuario y el programa imprima los 100 primeros múltiplos de ese número
   1. Mostrar la salida como contador cálculo
   2. Mostrar la salida como num\*con = calc
3. Escribir un programa en la cual al usuario se le pida dos números, siendo estos el primero el número del que quiere saber la tabla de multiplicar y el segundo cuántos valores de dicha tabla quiere. La salida ha de ser del tipo "7\*1=7"
4. Escribe un programa que reconozca una entrada de un número y nos diga si es negativo, positivo o cero.
5. Escribe un programa que reconozca la entrada de un número y nos diga si es positivo o negativo. Que pregunte constantemente hasta que el usuario introduzca un Cero.
6. Escribe un programa que compruebe si la última letra de una cadena es una A que suelte un mensaje.
7. Coger una cadena e imprimir todos los caracteres uno a uno de la cadena.
8. Mostrar una cadena por líneas mostrando en la primera línea el primer carácter, en la segunda línea el primer y segundo carácter, y así hasta la cadena final donde muestra la cadena completa.
   1. De 1 a Total
   2. De Total a 1
9. Imprimir los 100 primeros números en orden inverso. Del 100 al 1.
10. El usuario introduce dos números. El programa produce una serie de múltiplos del primer número de longitud al segundo. Ejemplo: 3,10 => 3, 6, 9, …. 30
11. El usuario introduce una cadena y un carácter, produciendo el programa como salida la cadena de entrada quitando los caracteres coincidentes con el introducido.
12. El usuario introduce una cadena y un carácter, produciendo el programa como salida la cadena de entrada invertida quitando los caracteres coincidentes con el introducido.
13. El usuario introduce una serie indefinida de números hasta que se introduzca un cero. El programa ha de imprimir cuántos números se han introducido, la suma de todos los números, cuántos de esos números son múltiplos de dos, cuántos son múltiplos de tres y cuántos son múltiplos de dos y de tres.
    1. Enunciado de arriba
    2. Reestructurar y usar una condición anidada, comprobando si es múltiplo de tres dentro de la comprobación de si es múltiplo de dos.
14. Escribir un programa que lea una serie de números y los imprima en orden inverso, dándonos también su media aritmética.
15. Escribir una función en la que introduciendo una cadena devuelve dicha cadena sin blancos. La función se llamará *sin\_blancos()*.
16. Escribe la función *invierte*. Al cual se le pasa una cadena y devuelve dicha cadena de forma invertida.
17. Escribir una función que lea una cadena desde teclado hasta que se introduzca una cadena vacía. Dicha función se llamará *leer*. Esta función quitará los blancos e invertirá la cadena, imprimiendo el resultado.
18. Escribir una función a la que se le pase una cadena, la cual devolverá verdadero o falso. La función es llamará *es\_palindromo*.
    1. Hacerlo usando las funciones del ejercicio 15 y 16.
    2. Hacerlo con un algoritmo propio.
19. Vamos a imprimir una tabla ASCII desde el 32 hasta el 255 (se puede probar desde 0, a ver qué hace).
20. Definir una función *codifica* a la que se le pasa una cadena de caracteres (mensaje) y un número. La función desplazará el carácter tanto como el número diga dentro de la tabla ASCII.
    1. Definir la función *codifica*
    2. Definir la función *decodifica*
21. Definir una función *rotar* a la que se le introduzca una cadena y un número. Esta función devuelve una cadena la cual es la rotación a la derecha N veces de la cadena de entrada.
22. Vamos a construir una pila LIFO *last in first out*. Vamos a crear unas funciones para manejar dicha pila. Estas funciones serán:
    1. *introduce(pila, x)* Añadimos un elemento al final de la pila
    2. *saca(pila)* Devolvemos el último elemento de la pila
    3. vacia(pila) Devuelve un boolean, true = pila vacía
23. Hacer el juego de las Torres de Hanoi basándonos en los métodos del Ejercicio 22.
    1. *mueve(x, y)* Mover el último elemento de la pila *x* a la pila *y*.  
       *La función mueve detecta si existe un disco de más peso sobre uno de menos y no permite el movimiento. Produce un mensaje de error.*
    2. *cima(x) Devuelve el último elemento de la pila sin borrarlo*
24. Dada una lista, queremos insertar detrás de cada elemento de dicha lista, el mismo elemento que estamos recorriendo.
25. Vamos a hacer una inserción ordenada en lista. Es decir, cada vez que se introduce un número en la lista, éste se insertará en la posición que le corresponda manteniendo el criterio de ordenación de la lista. De tal forma que la lista siempre está ordenada.
    1. El programa va a contener una lista de números indefinida, el cierre de función la hará la introducción de un 0 por parte del usuario.
    2. Definir la función *inserta\_ordenado(lista, elemento)*. Esta función NO devuelve nada.
26. Definir una función a la cual se le pase una cadena y ésta devuelva cuántas vocales tiene esa cadena.
    1. Usar *in* dentro de los *if* para la función.
    2. Definir una función *vocal* que devuelva un boolean dependiendo de si es vocal o no
27. Definir una función a la cual se le pase una cadena y ésta devuelva el número de ocurrencias de cada una de las vocales.
28. Queremos construir una lista por comprensión que contenga, de un texto dado, guarde los caracteres de la cadena según su posición
    1. Primera solución  
        'Hola Pepe' => ['H', 'Ho', 'Hol', …, 'Hola Pepe']
    2. Invertir las cadenas  
        'Hola Pepe' => ['H', 'oH', 'loH', …, 'epeP olaH']
    3. Construir una lista que contenga el tamaño de cada elemento de la lista de la solución *a*
29. Construir una lista por comprensión que contenga todos los múltiplos de tres hasta una cifra estipulada.
    1. Solución original
    2. Extender a incluir los números múltiplos de 5 o 7
    3. Incluir sólo los múltiplos de 3 y 7
30. Crear una lista por comprensión que contenga todas las sílabas posibles en castellano cuya primera letra sea una consonante y cuya segunda sea una vocal.
    1. Solución original
    2. Palabras de dos sílabas con todas las combinaciones creadas por la solución anterior
31. En una empresa hay 15 trabajadores. Se quiere hacer un torneo de ajedrez. Hay que escribir una función a la que se le pasa una lista de todos los empleados de la empresa y la función devuelve otra lista con todas las partidas posibles entre empleados. Nota: Un empleado no juega contra sí mismo.
    1. Solución Original
    2. Generar el orden en el que se van a ejecutar las partidas.  
       Generamos un número al azar del tamaño de la lista, sacamos la pareja elegida y la guardamos en una nueva lista, borrando la pareja de la original.
32. Crear una lista con 1000 números generados al azar entre 1 y 100.
33. Escribe una función a la que se suman dos puntos y se hace la suma vectorial de dichos puntos.
    1. Solución Original
    2. Definir la resta vectorial
    3. Definir el módulo del vector (distancia al origen)
34. Escribir una función a la que se le da como parámetro una tupla de números enteros y devuelva la suma de todos los elementos de dicha tupla.
    1. Solución Original
    2. Hacer otra función *resumen(notas)* a la que dada la tupla nos devuelva otra tupla que conste de tres elementos internos: cantidad de elementos, la suma de ellos y la media aritmética
    3. Generar entre 10 y 20 notas al azar para 40 alumnos distintos
    4. Definir una función que imprima el resumen de las notas de los alumnos
35. Hacer un conversor de gradación.
    1. De Celsius a Fahrenheit y de Fahrenheit a Celsius
    2. Escribir una función que genere aleatoriamente valores de temperatura entre -5 y 40ºC
    3. Escribir una función que genere una tupla con 1200 valores de temperatura
       1. Tupla ordenada, de forma creíble
       2. Tratar los datos de un fichero de la aemec
    4. Considerar que cada 100 datos del apartado C es un mes. Dar una media de temperaturas por meses
    5. Definir una función que traduzca la tupla de Celsius a Fahrenheit y viceversa
36. Modificar el ejercicio 20 para que el desplazamiento tome por defecto el valor por omisión 1 en el desplazamiento, además para la cadena se le pasará una sucesión indefinida de cadenas a codificar. La función devuelve una lista todas las cadenas codificadas.
37. Crear utilidades matemáticas:
    1. Escribir una función a la que se le pasa un número y devuelve una tupla con sus divisores.
    2. Se define un número primo como aquel que no tiene más divisores que él mismo y la unidad. Escribir una función que nos devuelva un True en caso de que ser un número primo.
    3. Crear una función a la que se le pasa un límite y nos devuelve una lista con todos los números primos por debajo de ese límite.
    4. Seguir el método de la Criba de Eratóstenes.
    5. Escribir una función a la que le vamos a pasar como parámetro un número que indicará una potencia de 10. Imprimirá la cantidad de primos y el porcentaje de números primos hasta el límite introducido.
    6. Escribir una función *segmento*s*\_primos(limite, ancho)* y devuelva una lista de tuplas que cuente el número de primos dentro de un rango que irá de *ancho* en *ancho* hasta *limite*.
38. Comprobar las Leyes de Demorgan
    1. not (AUB) = not A or not B
    2. not(AorB) = not A and not B
39. Definir los siguientes conjuntos (profesores de Bilingüe, FOL e Informática):  
     B = {'Ramón', 'Paco', 'Tessa'}  
     F = {'José Manuel'}  
     I = {'Paco', 'Belén', 'Blas', 'Manolo', 'Jose', 'Paqui', 'Ramón'  
    Definir dos más que sean Profesores y Profesoras.

A través de operaciones con conjuntos obtener

* 1. Todos los profesores NO Bilingües
  2. Profesores que sean de informática O de FOL
  3. Todos los profesores (que no profesoras) que NO sean de Bilingüe
  4. Todas las profesoras Bilingües
  5. Todos los profesores de Bilingüe o Informática, pero no incluídos
  6. Todos los profesores que NO sean Bilingües NI de FOL NI Mujeres

1. Buscar todos los números impares NO primos menores que mil y contarlos. Y cuántos números pares y primos hay.
2. Crear un diccionario que contenga una tupla binomial como clave, y como valor tendrá la distancia al origen del punto especificado por la tupla. Escribir una función a la que se le pasa un punto y devuelve la distancia al origen del punto.
   1. Solución Original
   2. Generar una nube de puntos al azar e incluirlas en el diccionario, dentro del rango [-100, 100]
3. Escribir un programa que lea un texto y devuelva un diccionario conteniendo en cada valor todas las palabras de una longitud determinada, siendo esta longitud la clave de dicho diccionario.

#### Glosario

* Código Fuente: Texto principal que contiene un programa.
* Constante o Literal: Es un valor que no cambia. Puede ser de cualquier tipo.
* Tipo:
  + <int> Integer 3
  + <float> Float 3.5  
    *Python no tiene límite en el tamaño numérico, no se necesita un BigInteger ni nada por el estilo.*
  + <str> String “Hola”  
     Character ‘x’  
    *En python las comillas simples y las comillas dobles son “lo mismo”.*
  + <bool> Boolean True/False  
    *Atento a las mayúsculas en la T y la F, ¡son necesarias!  
    Son un subtipo de los enteros.*
  + <list> Lista ['H', 'o', 1, 0.2]
* Variable: Referencia a un valor.  
  *Una variable toma su tipo una vez asignada, en caso de reasignación a otro tipo, el tipo de la variable cambiará. Aún y así Python mantiene un lenguaje estricto de tipos.  
  Es Case Sensitive, es decir, distingue mayúsculas de minúsculas.  
  El nombre del identificador ha de contar, al menos, con un carácter no numérico (letras no acentuadas o guión bajo). Y nunca ha de empezar con un número.  
  En Python no existen las variables privadas ni protegidas, sintácticamente hablando.*
* Referencia: Una dirección de memoria que apunta a un valor.
* Operadores:
  + Suma y Resta + -
  + Multiplicación y División \* /
  + División Entera //  
     *Devuelve el suelo de la división. Para entendernos Math.floor() en Java*
  + Potencia \*\* *Cada operador tiene un nivel de precedencia (o prioridad) de ejecución. Coincide con las precedencias matemáticas.*
  + Módulo %
  + Concatenación (String) +  
     *Si se van a concatenar dos Strings literales de forma directa, no es necesario el operador +  
     ‘abcdef’ <= ‘abc’ ‘def’*
  + *Concat Repetida (String) \*  
     ‘Hola’ \* 3 ‘HolaHolaHola’*
  + Comparación
    - Igual que == # *Igualdad NO es Identidad*
    - Menor o menor que < >
    - Men/May igual que <= >=
    - Distinto que !=
  + Operadores Booleanos
    - Y and
    - O or
    - NO not  
      *El orden lógico de operaciones es tal que not > and > or*
* Expresión: Combinación de constantes, variables y operadores.  
   x = (a + 3) \* 4  
   *La expresión ha de usar operandos del mismo tipo.  
   Como con todos los idiomas, primero se calcula la expresión y luego se asigna a la variable.*
* Asignación: Es una sentencia ejecutable con la que se le atribuye un valor a una variable.  
   variable = expresión  
   *En Python, si la variable ya existía, se destruye y se vuelve a crear, pudiendo ser de otro tipo distinto al anterior.  
   El símbolo ‘=’ NO significa que sirve para comprobar igualdad, es un símbolo de asignación.*
* Comentario: Es una anotación dentro del código fuente, no es ejecutable.
  + Comentario de Línea #
  + Comentario de Región ‘’’ ‘’’  
     *Se usará de forma obligatoria para documentación de código*
* Sentencia: Es una orden de ejecución al intérprete. Constituye una sola unidad de ejecución.
* Función: Se ejecuta mediante un identificador y sus argumentos dentro de paréntesis.  
   print(‘Hola’ ‘\n’ ‘¿Qué tal?’)
* Caracteres de Escape: Python reconoce los caracteres especiales dentro de las Strings.
  + Salto de Línea \n
  + Tabulación \t
* Tupla: Sucesión de elementos separados por coma. Suelen ir entre paréntesis, aunque no es condición indispensable.  
   *t = (1, 2, 3)*Son muy parecidas a las listas, con la condición de que son **inmutables**. También son más rápidas en ejecución que las listas, siendo más pequeñas en memoria que éstas. Permiten *indización* y *segmentación*. En caso de querer introducir un único elemento hay que especificar la coma:  
   *t = (1,)*Las tuplas se pueden *desempaquetar* (palabra mágica de Python). Asignación múltiple de elementos a una tupla.  
   *t = 1, 2, 3*  
   *a, b, c = t*  
   => *a = 1, b = 2, c = 3*Podemos usar una tupla como si de un registro se tratase (ejemplo: los nombres de columnas de una tabla de datos).  
  Usando el operador *\** podemos desempaquetar una tupla: *\*tupla*

##### Glosario de Funciones

Una función es un segmento de código que puede ser agrupado bajo un mismo nombre. Dicho segmento puede ser invocado desde otro punto cualquiera del código.

*function(arg1, arg2, arg3...)*

* print(String) Imprimir la String en pantalla
* input('prompt') Capturar el String del teclado, estaría bien asignarlo
* type(Object) Devuelve el tipo del Objeto (literal, variable o expresión)
* len(Object[]) Devuelve el tamaño de un Array (String o List)
* literal(Object) Convertir a tipo de literal
  + int(Object)
  + float(Object)
  + str(Object)
  + bool(Object)
  + list(Object)
* abs(Float) Toma el valor absoluto del float o integer introducido
* range(Integer) Crea una lista de N elementos comprendidos entre 0 y N-1
  + range(n, i) Crea una lista de I-N elementos comprendidos entre N e I-1
  + range(n, i, t) Lista entre N e I-1 saltando elementos de T en T  
    *SOLO ADMITE INTEGERS.*
* *ord(Char) Devuelve el código ASCII de un carácter.*
* *chr(Integer) Devuelve el carácter según un código ASCII.*
* *reversed(List) Devuelve un iterator invertido de la lista introducida.*
* *help(Function) Nos da una descripción de la documentación interna*
* *del(Object) Destruye cualquier variable en cualquier contexto*

###### Definir y llamar una función

*def function\_name(<params list>):* # NO se define el tipo de los parámetros  
 code  
 code  
 code  
 code

*def function\_name(a, b = 0)*

Suponiendo que sólo había dos parámetros, se hace una copia de la referencia de a y b. La función devuelve un valor y continúa la ejecución después de la llamada. La devolución se lleva a cabo a través de la sentencia *return*. También existen funciones procedimentales que realizan un trabajo pero no devuelven un valor asignado sino que devuelve *None* por cuenta propia.

El ámbito, o alcance, de los parámetros es desde y hasta dónde alcanza la función. Los parámetros de la función no se pueden usar fuera de ésta. Dos variables con el mismo nombre en diferente ámbito pueden coexistir en la misma ejecución. Una variable solo funciona dentro de su ámbito.

OJO: Los valores por omisión (*b = 0*) se definen sólo al definir la función en la "compilación", no en cada llamada. Si el Valor por Omisión es un objeto mutable, éste se modificará ante las siguientes llamadas de la función. Vamos, que es más bonito usarlo para constantes, para no liar la marrana sin querer.

*def f(a, L=[])*

*L.append(a)*

*return L*

*Cada vez que se llame a la función de arriba NO será una lista vacía la que se use, sino la misma lista siempre.*

Se puede asignar un parámetro cuando haces la llamada a la función:

*def saluda(persona, texto='Hola'):*

*print (texto, persona)*

*saluda(persona='Pepe')*

Para tener una función multi-parámetros (no definidos) usamos el método de desempaquetado de tuplas:

*def f(a, b, \*args):*

*""" Esta función tiene dos parámetros obligatorios posicionales (a y b)*

*El resto de argumentos a partir de ahí son opcionales e irán  
 empaquetados en una tupla*

*"""*

*print(a, b, 'Resto:', \*args)*

##### Normas de Estilo

* Los nombres de variables, funciones, clases, etcétera han de ser significativos (las ofuscaciones que las haga algún programita automáticamente después).
* Los nombres de variables empiezan siempre con minúsculas y se pueden separar usando CamelCase o guión bajo.  
   *nombreVariable nombre\_variable*
* Como las variables privadas no existen, su nombre se inicia con un guión bajo para no usarlas cuando no toque.
* Los nombres de las clases empezarán con mayúsculas y se separará mediante CamelCase.
* Los nombres de constantes estarán escritos en mayúsculas y se separarán con guión bajo.
* Para el sistema de indexación vamos a (muy a mi pesar) mantener el uso de 4 espacios, habría que asegurarse de que el editor de texto tiene dicha configuración. En su defecto usar tabulaciones puras. La cantidad de 'blancos' es importante.
* Cuando escribamos una función que devuelva un valor, el código de la función debe ser estrictamente el necesario para calcular el valor a devolver. Nada más. NO deben realizar ninguna petición de entrada ni de salida.
* Los nombres de los identificadores de funciones se escribirán en minúsculas, separando las palabras con guiones bajos. Al igual que los nombres de los parámetros. Han de ser significativos.  
   nombre\_funcion
* La definición de las funciones se hará después de la inicialización de variables.
* La longitud de las líneas de los comentarios ha de ser de unos 80 caracteres.
* Hay que separar las declaraciones de funciones y clases con saltos de línea.
* Los comentarios de final de línea son convenientes, pero sin abusar.
* Usar un blanco delante y detrás de los operadores.
* La codificación de caracteres en Python ha de ser en UTF-8.
* [PEPs - Python Enhancement Proposals](https://www.python.org/dev/peps/)
* No poner espacios alrededor de los índices ni los parámetros.
* Poner un blanco después de una coma en las listas.
* Siempre que vayamos a recorrer una lista con un for, usaremos una copia de dicha lista usando bien el método *copy()* on*[:]*.
* Los programas han de estar Factorizados. Se llama factorizar a definir diversas funciones que realicen partes de trabajo de un código sustituyéndolo luego en éste. Mejor tener muchas funciones que un código espagueti. Gracias a esto no sólo podemos leer y depurar el código más fácilmente, sino que nuestro código será reutilizable.
* Evitar *prints* de variables muy largas.
* Los *import* irán debajo de la documentación inicial, al principio del archivo.

##### Sentencias de Control de Flujo

Son sentencias que cambian la ejecución secuencial del programa, permitiendo seguir otros "caminos" a la hora de ejecutar.

No tienen delimitador sintáctico (no hay {} como en otros lenguajes), funciona por indexaciones (como los .yml).

* if <condition>: # condition = boolean  
   code  
   code  
  elif <condition>: # condition = boolean, mismo que elseif  
   code  
   code  
  else: # en caso de que no se cumpla nada de lo anterior  
   code  
   code
* while <condition>: # condition = boolean, estructura repetitiva por condición   
   code # en caso de bucle infinito por mala condición usar [ctrl + c]  
   code
* for <elem> in <list>: # itera sobre una secuencia, ideal que list sea copia  
   code  
   code
* pass # no hace nada, preparar un "TODO" que no rompa el código

##### Listas

###### Métodos

*Presuponemos que nuestra variable List se llama lista:*

* lista.append(Object) Agregar un objeto al final de la lista
* lista.extend(List) Concatena una lista a la primera
* lista.insert(i, x) Insertar el objeto *x* en la posición *i*
* lista.remove(x) Elimina la primera ocurrencia de *x* en la lista
* lista.pop() Elimina y devuelve el último elemento de la lista
* lista.pop(i) Elimina y devuelve el elemento de la posición *i*
* lista.clear() Limpia la lista, dejándola vacía.
* lista.index(x) Devuelve el índice de la primera ocurrencia de *x*
* lista.count(x) Devuelve el número de veces que aparece *x*
* lista.sort() Ordena una lista siguiendo su orden natural
* lista.reverse() Invierte el orden de los elementos de la lista
* lista.copy() Devuelve una nueva lista copia de la primera

***Listas por Comprensión***

Una lista por comprensión es una forma de construir una lista de contenido definido a partir de una expresión. Es equivalente a uno o varios *for* anidados.

lista = [expresion for… for… if...]

*lista = [x for x in range(1000)]*

En la parte izquierda del corchete se especifica qué es lo que se va a incluir en la lista, y a su derecha irán los *for* e *if* necesarios para condicionar el elemento a incluir.

##### Set

Viene a ser un conjunto de elementos no ordenados ni repetidos. Para declararlo se hace tal que:

*variable = set()*

Para comprobar si un elemento está dentro de un set usamos:

*if elemento in variable:*

###### Operadores

* Unión | Los elementos de ambos conjuntos.
* Intersección & Los elementos comunes entre ambos conjuntos.
* Exclusión - Los elementos de uno eliminando la intersección.
* Unión Excluída ^ Los elementos de ambos exceptuando la intersección

##### 

##### 

##### 

##### Diccionarios

Un diccionario es una estructura similar a una lista pero a la cual se accede mediante claves. (Parecido a un *Map* en Java). Una clave será un valor de cualquier tipo que sea INMUTABLE, en el valor se admite cualquier tipo, inmutable o no.

Inicialización vacía de un diccionario:

*dic = dict()*

*dic = {}*

Introducir un valor en una clave:

*dic['B'] = 25*

Inicializar un diccionario con valores:

*dict = { K1:V1, K2:V2, K3:V3 … }*

Por lo que se puede ver, un diccionario es un conjunto de pares Clave:Valor. Las claves son ÚNICAS.

***Métodos***

* *.keys() Devuelve todas las claves del diccionario*
* *.values() Devuelve todos los valores del diccionario*
* *.clear() Limpia el diccionario (lo deja vacío)*
* *.copy() Devuelve una copia del diccionario*
* *.get(key) Devuelve el valor de la clave key*
* *.delete(key) Elimina el par [clave, valor] del diccionario*
* *.items() Devuelve una lista [clave, valor] del diccionario*
* *.pop(key) Devuelve y elimina el par [clave, valor] del diccionario*

##### 

##### 

##### Sistema de Paquetes

Para importar el módulo de un paquete usamos *import*.

*import Package.Module*

Es posible importar sólo una función de un módulo usando *from*.

*from Package.Module import Function*

Cuando se hace una importación, sea de un módulo o una función, en el cuadro de nombres principal se agregan los de dicho módulo/función, siendo reconocibles por nuestro programa.

###### Módulos Propios de Python

* *random* Paquete de generación de aleatorios
  + *randint(x,y)* Genera un entero al azar entre dos valores, ambos incluídos
* math
  + *sqrt(x)* Devuelve la raíz cuadrada del número *X*
  + *sin(x)* Devuelve el seno de *X*
  + *cos(x)* Devuelve el coseno de *X*
  + *tan(x)* Devuelve la tangente de *X*
  + *hypot(x, y)* Devuelve la hipotenusa de los valores *X* e *Y*
* *time*
  + *time()*

1. Siempre y cuando dicho sistema operativo cuente con su intérprete de lenguaje instalado. [↑](#footnote-ref-0)