SEAS
CAMPUS SEAS

1 UNIDAD DIDACTICA

Python

1. Primeros conceptos

ÍNDICE

OBJETIVOS

- En esta primera unidad se van a repasar someramente algunos conceptos básicos de informática al objeto de situar al alumno en ese contexto.
- Se verán también algunos conceptos básicos relativos a Python y, sobre todo, se centrará en una parte práctica que consistirá en la descarga de Anaconda, el intérprete de Python que se va a usar en el curso y del IDE PyCharm que se usará para escribir y gestionar los programas que se van a desarrollar a lo largo del curso.
- Finalmente se realizarán algunas operaciones elementales usando el IDE en modo calculadora y utilizando asignación de valores a variables y asignaciones externas desde el programa.

INTRODUCCIÓN

En esta primera unidad se van a ver algunos conceptos básicos de informática, tales como las partes principales de un ordenador tanto a nivel hardware como software, así como algunos hitos que han contribuido al desarrollo de la informática y la digitalización.

El objetivo de este curso es escribir programas en Python, y se verán algunas ideas acerca de cómo, un programa escrito en un lenguaje de alto nivel como Python, lo puede ejecutar una máquina que solo trabaja con unos y ceros.

De cara a la programación en esta unidad veremos el vocabulario empleado en Python, la declaración de constantes y variables, y el operador de asignación interna por el que toman valores las variables usadas en un programa. También la salida estándar de Python para poder escribir valores en la pantalla del ordenador.

1.1. Primeros conceptos

Vamos a comenzar con una definición genérica de **ordenador**.

DEFINICIÓN

Ordenador es una máquina programable para el tratamiento de la información. Por tanto:

- Su comportamiento viene caracterizado por la ejecución de un programa que reside en su memoria. El programa lo han concebido, diseñado y escrito un grupo de personas. Decimos que se trata de una máquina de propósito general.
- Resuelve problemas de tratamiento de la información: cálculo numérico, control automático, tratamiento de señales, simulación de procesos, gestión administrativa y de recursos, procesos estadísticos, etc. de amplia utilidad en casi todos los ámbitos de la sociedad.

La máquina está compuesta por la circuitería electrónica (el *Hardware*) y por los programas necesarios para su funcionamiento (el *Software*). Hay un *software* que reside permanentemente en memoria y su contenido no se puede cambiar. Se le suele llamar *firmware* (hecho firme).

Los componentes básicos de todo ordenador, son:

- Unidad Central de Proceso (CPU *Central Processing Unit*), encargada de realizar todo el conjunto de operaciones tanto aritméticas como lógicas y de controlar la secuencia correspondiente a las distintas acciones que tienen lugar cuando se ejecuta un programa.

- Memoria central. En la arquitectura de tipo Von Neuman, la memoria es el elemento que se encarga de almacenar los datos en binario (unos y ceros) con los que opera la máquina así como los programas. Es una memoria de lectura y escritura (RAM, *Random Acces Memory*).

- Periféricos. Encargados de comunicar al ordenador con el exterior. Se trata de pantallas, teclados, ratones, altavoces, impresoras, discos, sistemas de adquisición de datos, tarjetas de red, etc. Los programas de gestión de los periféricos se denominan "*drivers*" y deben estar instalados para que el periférico funcione correctamente (muchos drivers se pueden descargar por Internet).

La ejecución de un trabajo no elemental no es inmediata en ningún campo de la ciencia; tampoco en programación. Habrá distintas etapas (operaciones) que conducen hacia el fin deseado. A cada una de estas etapas elementales se le denomina una "acción".

Para una máquina dada, una acción es "primitiva" si dicha acción la sabe ejecutar la máquina. Una acción primitiva también se llama una sentencia o instrucción. Una acción no primitiva se dice que es "compuesta". Se llama secuencia al conjunto de sentencias que resuelven la acción compuesta.

En programación imperativa, un algoritmo es el enunciado de una secuencia (sucesión) de sentencias, que realizan el tratamiento necesario para la resolución de un cierto problema en un número finito de pasos.

Lo que se acaba de exponer corresponde al modelo de programación imperativa que es el más extendido. Quizás esto pueda inducir a los estudiantes a identificar el concepto de programa con el de una secuencia de sentencias. Sin embargo, existen otros modelos de programación, entre los que se encuentra la programación funcional, programación lógica, etc. que se salen fuera del ámbito del presente curso.

Dentro del software toman una gran importancia un conjunto de programas de *gestión de la máquina* que realizan una serie de funciones engorrosas que no debe realizar el usuario. Se trata del *sistema operativo*.

En los primeros tiempos de los ordenadores, los programas de aplicación realizaban el control de toda la máquina de forma que el programa podía usar todos los recursos hardware. No existía en la máquina ningún otro software al mismo tiempo. Esto implicaba que el programa debía realizar absolutamente todo a partir de cero, incluso los servicios básicos de entradas/salidas, gestión de memoria, etc. lo que complicaba mucho su programación.

Posteriormente se implementaron algunas utilidades en un programa especial que estaba siempre disponible, el sistema operativo (OS, *Operanting System*). Así, un programa de aplicación no necesitaba tener el control completo de la máquina, sino que iba a cooperar con el sistema operativo usando sus servicios donde fuese necesario.

En la década de los sesenta se probaron los SO de *tiempo compartido*. En tiempo compartido los usuarios acceden al ordenador a través de periféricos (normalmente teclado y pantalla). En tal sistema pueden haber muchos usuarios compartiendo aparentemente el ordenador al mismo tiempo.

El objetivo de los SO es presentar el computador al usuario y al programador en un cierto nivel de abstracción, ocultando el trabajo realizado en niveles inferiores. Por ejemplo, gracias al SO:

- La memoria se presenta en términos de variables de determinado tipo. Su gestión a nivel de máquina es transparente al usuario de forma que el usuario se despreocupa de en qué región de memoria está trabajando y qué región de memoria está libre para poder trabajar.
- Los dispositivos de memoria no volátil, tanto interna como externa, como el disco duro y los *pen-drive* se presentan en términos de "ficheros" (también "archivos").
- La pantalla como áreas rectangulares llamadas "ventanas" (*windows*), que pueden contener "vistas" (*views*), menús, *toolboxes*, etc.
- El teclado se ve como una secuencia de caracteres de entrada (entrada secuencial, un carácter tras otro).
- El ratón, como un par de coordenadas y el estado de un conjunto de pulsadores.

La tarea del sistema operativo es implementar y gestionar las operaciones anteriores, de forma que el usuario no tenga necesidad de conocer lo que sucede a nivel físico en la máquina. Para que se entienda más fácilmente, cualquier persona sabe utilizar un teléfono móvil. Sin embargo, la mayor parte desconoce su funcionamiento y su circuitería. Hoy en día los sistemas operativos de más uso son: Windows (en sus distintas versiones), Unix (en sus distintas variantes: SunOS, Linux, etc), MacOS, etc.

1.2. Hitos principales en la informática

Enumerar los principales hitos de la informática resulta complicado ya que constantemente se están desarrollando nuevas técnicas y aplicaciones, aunque puedan no tener la entidad suficiente como para considerar que corresponden a hitos en su evolución. Sin embargo, aún a costa de ser parciales, a continuación se presenta una pincelada de algunos de esos hitos.

En 1977 Apple Computer popularizó el fenómeno de los ordenadores personales con el Apple II. El precio de los ordenadores bajó lo suficiente para que muchas personas pudiesen tener un ordenador. Cuatro años más tarde, IBM, el mayor fabricante del mundo en la época, introduce el ordenador personal (PC, *Personal Computer*). A partir de ahí, ha habido un desarrollo sin precedentes en la industria informática, tanto a nivel *hardware* como *software*.

Sin embargo, estos ordenadores eran entidades autónomas; la gente hacía su trabajo en su propia máquina y luego transportaba discos de un lado para otro. Estas máquinas podían vincularse entre sí para formar redes de ordenadores, a través de líneas telefónicas o en redes de área local (LAN). Esto dio lugar a los **sistemas distribuidos,** en los que la computación se distribuye a través de redes.

Las máquinas más potentes de escritorio, llamadas *estaciones de trabajo* (*host*) ofrecen grandes prestaciones. La información se comparte a través de redes de ordenadores. Algunos de tales ordenadores llamados *servidores de archivos* ofrecen un almacén común para programas y datos que pueden utilizar otros ordenadores clientes distribuidos por la red. De ahí el término *cliente/servidor.*

Los primeros lenguajes de programación tales como el Fortran y el Basic pronto dieron paso a lenguajes estructurados. C++ es el lenguaje de programación preferido para escribir *software* de sistemas operativos, redes de ordenadores y aplicaciones cliente/servidor distribuidas. Posteriormente Java se convirtió en el lenguaje preferido para crear aplicaciones basadas en Internet y también, cómo no, para redes de ordenadores y aplicaciones cliente/servidor. Actualmente está cobrando un amplio consenso Python por su potencia y sencillez y se está usando en multitud de aplicaciones. El lenguaje de programación Python fue desarrollado por Guido van Rossum en 1989 y a lo largo del tiempo se ha beneficiado de numerosas contribuciones. De entre sus características cabe reseñar que es gratuito, portable, dinámico, modular y orientado a objetos.

Internet es una red de escala mundial que utiliza la familia de protocolos TCP/IP lo cual garantiza que las distintas redes físicas que la componen formen una red lógica única de alcance mundial. Sus orígenes se remontan a 1969 cuando se estableció la primera conexión llamada ARPANET. De 1990 data uno de los servicios mas populares de Internet, el *Word Wide Web* (WWW), que es un conjunto de protocolos que permite la consulta remota de archivos de hipertexto. La computación en la nube es un paradigma que permite ofrecer servicios de computación normalmente a través de Internet.

Internet de las cosas (IoT, *Internet of Things*) ha extendido la conectividad de Internet a todo tipo de dispositivos sensores para conectar entre sí multitud de sistemas de adquisición de datos. Esto provoca la existencia de grandes volúmenes de información (datos). El *Big Data* hace referencia a conjuntos de datos tan grandes que hacen falta aplicaciones no tradicionales para tratarlos adecuadamente.

1.3. Programacion

Los ordenadores, tal y como se usan hoy en día, no tienen inteligencia innata. Cualquier comportamiento aparentemente inteligente que exhiba el ordenador es consecuencia del programa que reside en su memoria. Ese programa lo han concebido, diseñado y escrito las personas. Lo amigable que sea un programa al usuario, y lo útil o lo frustrante que sea, está determinado en gran parte por la calidad de su diseño y por el lenguaje de programación utilizado. El alcance de la utilización de los ordenadores es tan grande que la industria del *software* ha llegado a ser multibillonaria en dólares.

1.4. Características del lenguaje Python

Aunque muchas de ellas no se comprendan ahora, vamos a hacer una relación de las características más destacadas de Python.

- Portable sobre las diferentes variantes de Unix, MacOS, y las diferentes versiones de Windows. Un compilador nuevo llamado *Jython* escrito en Java genera *bytecode* Java.
- Es gratuito y no tiene restricciones en proyectos comerciales.
- Trabaja igualmente bien con *scripts* de una docena de líneas que con proyectos complejos de decenas de miles de líneas.
- Su sintaxis es muy simple y combinada con tipos de datos evolucionados (listas, diccionarios, etc.) da lugar a programas muy compactos y legibles. A igualdad de funcionalidad un programa en Python es de 3 a 5 veces más corto que un programa en C, C++ o Java. Esto implica un tiempo de desarrollo mucho más corto y un mejor mantenimiento.
- Python genera sus recursos (memoria, descriptores de ficheros, etc.) sin intervención del programador, mediante un mecanismo de contaje de referencias (parecido, aunque distinto de un colector de basura)
- No hay punteros explícitos.
- Es opcionalmente multi-hilado.
- Es orientado a objetos, soporta herencia múltiple y sobrecarga de operadores. Toma la terminología de C++ y todos los métodos son virtuales.
- Al igual que Java, soporta un sistema de excepciones que permite simplificar considerablemente la gestión de errores.

- Es dinámico (el intérprete puede evaluar cadenas de caracteres representando expresiones o instrucciones). Es ortogonal (un número reducido de conceptos es suficiente para engendrar construcciones muy ricas). Es reflectivo (soporta la metaprogramación, por ejemplo, la capacidad para un objeto de añadir o eliminarse de los atributos o de los métodos o incluso de cambiar la clase en tiempo de ejecución. Es introspectivo (un gran número de herramientas de desarrollo implantadas en Python directamente como el *debugger* o el *profiler*).

- Tiene tipos dinámicos, es decir, todo objeto posee un tipo bien definido en tiempo de ejecución que no tiene necesidad de haber sido declarado anteriormente.

- Python es extensible y puede fácilmente trabajar con bibliotecas existentes de C. Puede servir como lenguaje de extensión para sistemas lógicos complejos.

- La biblioteca estándar de Python y los paquetes aportados dan acceso a una gran variedad de servicios: cadenas de caracteres y expresiones regulares, servicios UNIX estándar (ficheros, *pipes*, señales, *sockets*, hilos, …), protocolos de Internet (Web, News, FTP, CGI, HTML, …), persistencia y bases de datos, interfaces gráficas.

- Es un lenguaje que continúa evolucionando sostenido por una comunidad de usuarios. Paralelamente al intérprete principal escrito en C y mantenido por el creador del lenguaje, está en fase de desarrollo un segundo intérprete escrito en Java.

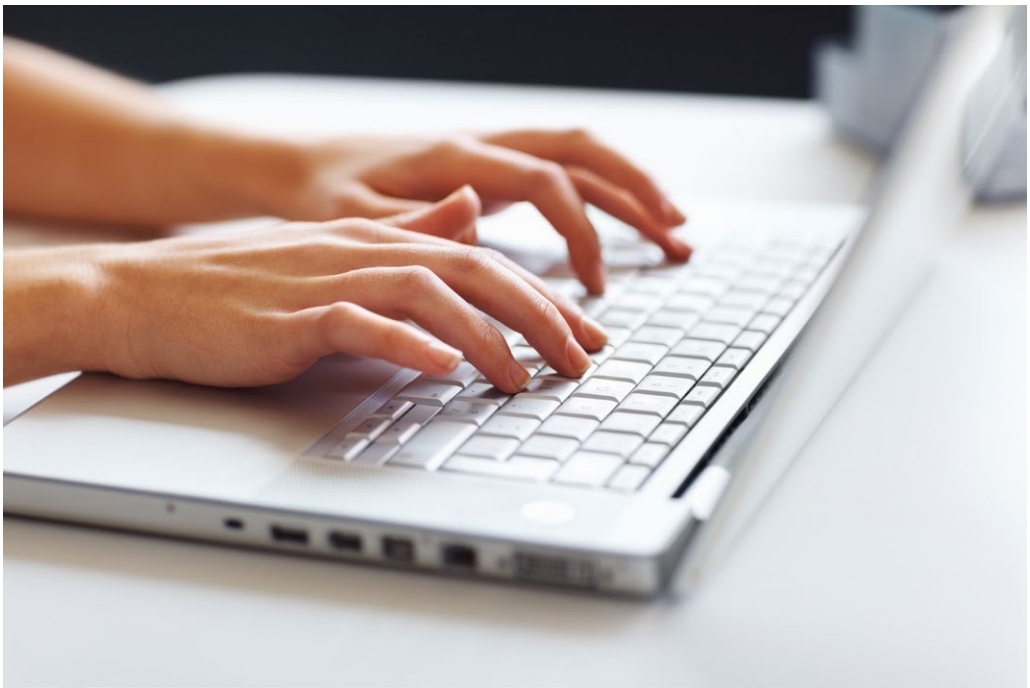
- Es un lenguaje de referencia para tratar XML.

1.5. Realización de programas en lenguajes imperativos

En primer lugar habrá que tener claras las ideas acerca de lo que debe realizar un programa para un determinado cliente que posiblemente no sabe bien qué es lo que quiere. Se necesitará una buena especificación del programa (que establezca lo más detalladamente posible la función que va a tener tal programa) y que sea lo menos ambigua posible, para lo que se necesitarán frecuentes cambios de impresión iniciales con el cliente.

Una vez que se tiene claro lo que se requiere del programa, la dificultad fundamental de la programación reside en la necesidad de pasar de nuestras ideas a un programa que pueda procesar una máquina. Se trata de una dificultad bastante parecida a la forma en la que nuestra mente resuelve los problemas, que requiere una cierta capacidad de abstracción, a lo que hay que añadir el hecho de que en el caso de programación en gran escala el problema lo tendrá que resolver un grupo de programadores.

Formalmente el proceso mental para escribir un programa es similar al que se lleva a cabo para resolver cualquier problema. La forma de pensar para escribir un programa cambiará según se escriba en programación imperativa o, por el contrario, se aborde mediante programación orientada a objetos. En el primer caso, se intentará descomponer un problema en problemas de menor entidad (diseño descendente), más sencillos de resolver, hasta llegar a una solución que pueda procesar una máquina. En el segundo caso, nos interesaremos por los objetos que va a manejar nuestro programa y veremos qué acciones se pueden llevar a cabo con dichos objetos. Como se verá posteriormente, pese a ser orientado a objetos, Python permite distintas formas de programación.



1.6. Programas fuente y objeto. *Bytecode*

El procesador de un ordenador únicamente sabe ejecutar sentencias escritas en lenguaje de máquina, es decir, en un conjunto de unos y ceros agrupados en palabras que constituyen el programa, y que deben residir en la memoria.

Cuando los programas son complejos, resulta muy difícil escribirlos en lenguaje de máquina, por lo que se escriben en lenguajes de alto nivel, como Python C++ o Java entre otros muchos de los lenguajes existentes. Pero el procesador no sabe ejecutar un programa escrito en un lenguaje de alto nivel. Por esta razón, a partir de un programa escrito en estos lenguajes, hay que transformarlo creando un programa en lenguaje de máquina que el procesador sepa ejecutar. Esa es la función del programa traductor (un compilador, un intérprete o mezcla de ambos) que es un programa que se encuentra almacenado en un fichero. La diferencia entre un intérprete y un compilador es que el intérprete traduce instrucción a instrucción y ejecuta cada instrucción una vez que está en lenguaje de máquina. Por el contrario, el compilador genera un fichero con todo el programa traducido y posteriormente lo ejecuta.

En lenguaje C, cuando se compila un programa, se produce otro programa en lenguaje de máquina, que el procesador ya sabe ejecutar. El compilador ha creado un programa para ese procesador. Esto genera una dependencia del procesador que tenga la máquina, por lo que el código no es portable en otras máquinas distintas.

- Programa fuente. Al programa escrito en el lenguaje de alto nivel se le llama "programa fuente". En Python, el código fuente que se ha escrito se almacena en un fichero de extensión ".py". En Java, el programa fuente se almacena en un fichero de extensión ".java". En C, la extensión sería "c", mientras que en C++ bajo Windows sería ".cpp", bajo Unix sería ".cc" etc.

- Programa objeto. Al programa que resulta de la compilación se le llama "programa objeto" y se almacena en un fichero de código. En las figuras 1.1 y 1.2 se muestran estas ideas. En C y C++ el fichero que contiene el programa objeto tiene una extensión ".o" (*inicial de object*).

Figura 1.1. Representación gráfica de los programas fuente y objeto.

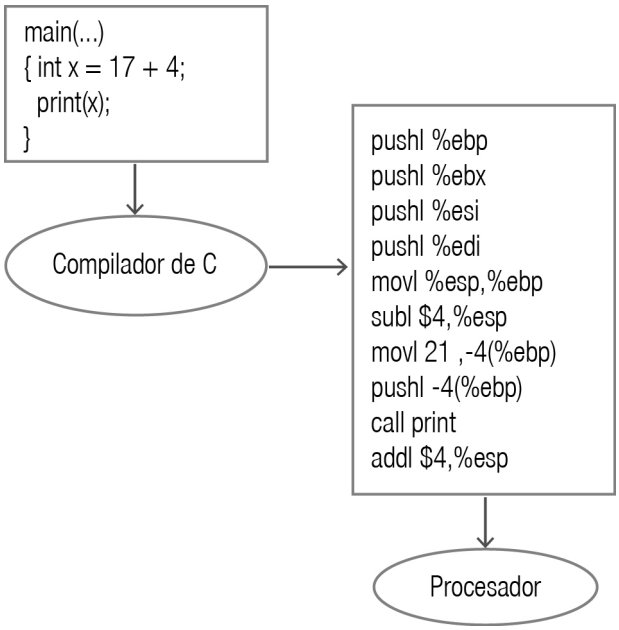


Figura 1.2. Programa compilado.

Veamos lo que sucede en Java con el programa objeto. En Java, se ha desarrollado un procesador virtual (se llama así porque en realidad tal procesador no existe en la máquina). Recibe el nombre de "máquina virtual Java" (JVM, *Java Virtual Machine*) y tiene sus propias instrucciones que sabe ejecutar. Entonces, el resultado de la compilación de un programa es un archivo (de extensión .class) que no corresponde al lenguaje de máquina del procesador que tenga el ordenador. Decimos que es un archivo de "*bytecode*" (se traduce por: códigos de *bytes*). Este procesador es único y universal, por lo que se generan los mismos *bytecode* independientemente del procesador y del sistema operativo.

Para que una máquina concreta pueda ejecutar esos *bytecode*, necesita de un navegador de Internet, que los interpreta y ejecuta (para ello hay que crear un archivo de HTML que usará el navegador para ejecutar el *applet*). Con el *software* de Java se incluye un visor de Applets para poder ejecutarlos sin necesidad de estar conectados a Internet. En las figuras 1.3 y 1.4 se muestra el modelo de Java:

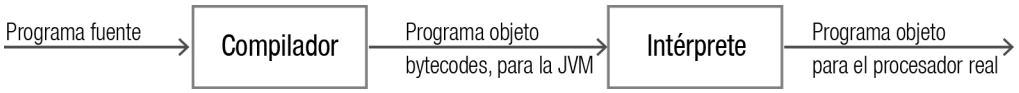


Figura 1.3. Máquina virtual Java.

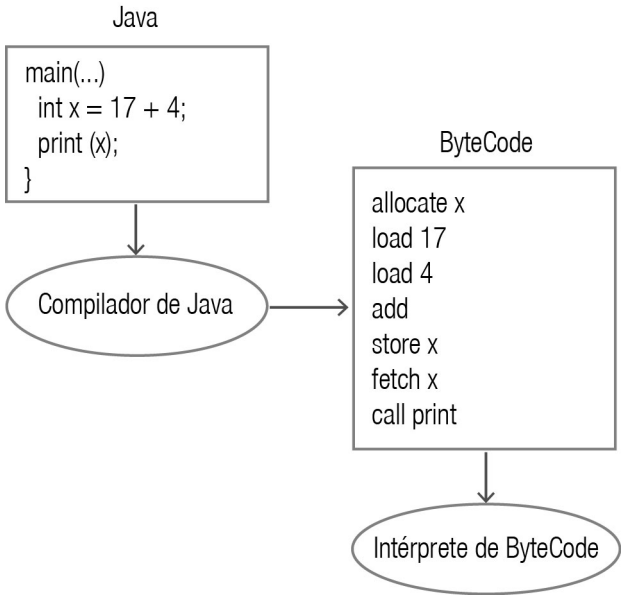


Figura 1.4. Máquina virtual Java mostrando los bytecode.

Python también trabaja con *bytecode*. Como se ha dicho, el fichero de extensión ".py" contiene el código fuente que se ha escrito. El fichero compilado de *bytecode* tiene por extensión ".pyc". Caso de que estén activadas las optimizaciones el archivo que se genera tiene por extensión ".pyo".

Python puede estar implementado de varias maneras. Una de ellas es CPython, que está escrita en lenguaje C, que compila el código escrito en Python a código *bytecode* intermedio que posteriormente interpreta la máquina virtual. Otra implementación es Jython que compila el código escrito en Python en bytecode de Java. Existen otras implementaciones que se salen del ámbito de esta introducción.

Una de las características notables de Java y Python es su capacidad multiplataforma. De todos los entornos de desarrollo de *software* que hay actualmente Python es el que mejor permite escribir programas complejos que trabajarán (salvo quizás pequeñas modificaciones) en alguno de los anteriores sistemas operativos.

1.7. Vocabulario y representación

- Literal: un *literal* denota un valor constante. Este valor puede ser numérico (entero o real), carácter, booleano ("*True*" o "*False*") o cadena de caracteres. Python también tiene literales especiales como "*None*" y Colecciones.

- Dígito: un dígito es un número del 0 al 9.

- Símbolos: la representación de símbolos en términos de caracteres se define usando la norma ISO 8859-1. Los símbolos pueden ser: identificadores, números, cadenas de caracteres, operadores y delimitadores.

En Python se van a observar las siguientes reglas:

- Los símbolos no deben llevar espacios en blanco ni guiones, salvo naturalmente en los comentarios y espacios en blanco en las cadenas. Las mayúsculas y las minúsculas se considerarán DISTINTAS.

- Un literal carácter (o cadena de caracteres) se expresa encerrando el carácter entre comillas dobles " " o simples ' '.

1. Identificadores. Todo nombre que se ponga en un programa se conoce como identificador. Son secuencias de letras, dígitos y el símbolo de subrayado (*underscore*) introducido en Python para una mayor semejanza con los lenguajes C++ y Java. El primer carácter no debe ser un dígito. Consideraremos que nuestros lenguajes son tales que distinguen las mayúsculas de las minúsculas, por lo que mayúsculas y minúsculas son diferentes.

   Existen ciertas palabras reservadas que pertenecen al lenguaje y no se pueden usar como identificadores.

   En Python no puedes empezar (ni usar en) el nombre de variable usando los signos aritméticos: *, +, -, /

   Tampoco usando símbolos de aritmética avanzada: ^, &, %

   Tampoco puedes usar . ni , ni tampoco ; = : ' " ?

   Tampoco puedes usar números al inicio del identificador.

   La recomendación es no usar palabras que tiendan a confusión.

   Entonces el símbolo de subrayado _ sí está permitido y puedes usarlo incluso para iniciar el nombre de los identificadores (tanto en Python como en muchos otros lenguajes)

   No puedes usar palabras reservadas. Pero sí puedes usar palabras como: integer, floating y otras que, aunque parecen similares, no son iguales a las reservadas.

2. **Números**. Son enteros (llamados *int*), reales (llamados *float*) y complejos (llamados *complex*). El signo de un número se introducirá mediante un carácter especial (+, -) que precede al número, aunque el signo positivo no es necesario escribirlo ya que se considera implícitamente cuando no hay signo.

```
>>> e1=7
```

Los valores literales enteros pueden expresarse en otros sistemas de numeración distintos del decimal. Por ejemplo, en hexadecimal (base 16) u octal (base 8). Un número hexadecimal debe ir precedido de los caracteres "0x". Para escribir un número octal, el número debe ir precedido del carácter "0".

Decimal	Octal	Hexadecimal
8	0o10	0x8
16	0o20	0xa
90	0o132	0x5a

Los números reales (*float*) siempre tienen punto decimal. Más adelante se estudiarán a fondo las representaciones numéricas y el resto de los símbolos.

3. **Literales carácter**. Un literal carácter es un carácter encerrado entre comillas simples o dobles.

```
>>> c1 = 'c'
```

```
>>> c2="C"
```

4. Literales **booleanos**. En Python se denotan mediante *True* (verdadero) o *False* (falso).

5. **Cadenas de caracteres**. Son sucesiones de caracteres. Existen cadenas cortas y largas. Un literal de cadena corta es una sucesión de caracteres encerrada entre comillas dobles o simples. Por ejemplo:

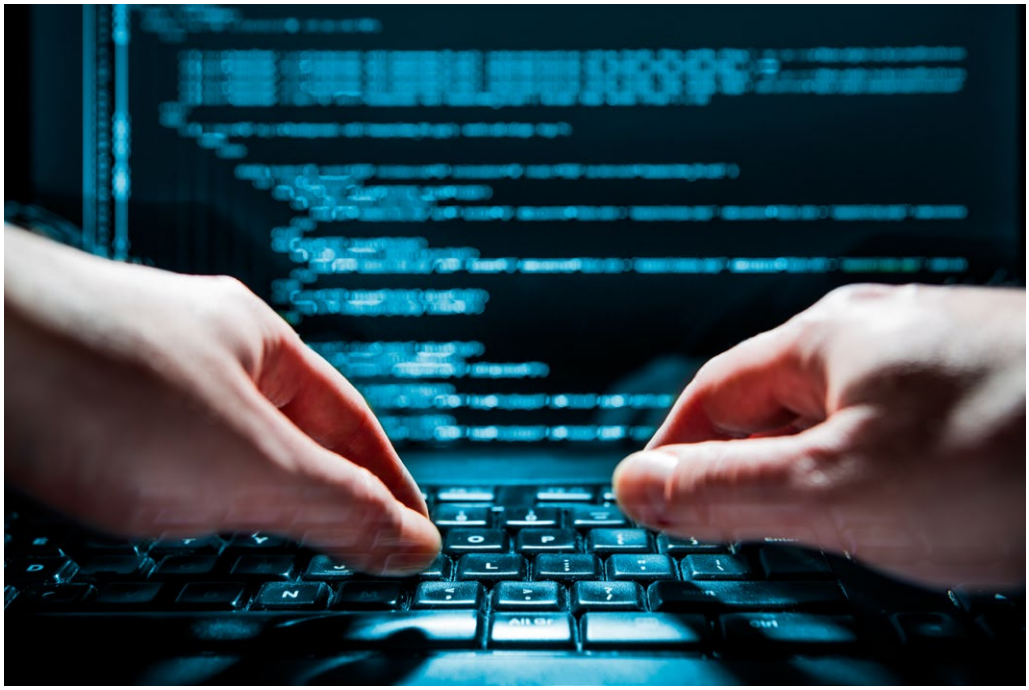
```
>>> 'Hola'
```

Un literal de cadena larga corresponde a caracteres encerrados entre comillas triples. Por ejemplo:

```
>>> '''Hoy me voy de viaje en tren, mañana en avión'''
```

1.8. Declaración de constantes

Como ya sabemos, un algoritmo trata la información contenida en forma de datos. Los datos representan una abstracción de la realidad y constituyen una representación asociada a un determinado fenómeno. Así, pueden representar valores de una señal de vibración de una máquina, una señal de electrocardiografía, las ventas de una empresa, los impuestos, presiones, temperaturas, señales de control, etc. Procesar los datos es transformarlos en una representación más útil al usuario.



El dato puede ser constante si siempre conserva el mismo valor mientras se ejecuta el algoritmo. Modificar su valor implica modificar el programa fuente y volver a ejecutarlo. Será variable si puede variar su valor mientras se ejecuta el algoritmo.

Aunque no sería necesario el empleo de constantes, sí que resulta conveniente su uso, de cara a realizar un mantenimiento más sencillo de los programas. Por ejemplo, si se declara una constante *iva* que tome por valor 21, será más sencillo modificar su valor si *iva* se ha declarado como constante, que si hay que mirar a lo largo del programa en dónde aparece el número 21.

En Python las constantes se asignan y se declaran en un módulo. Más adelante veremos el concepto del módulo. En este momento significa un fichero que contiene objetos informáticos como constantes, variables, funciones, etc. Las constantes se suelen escribir en letras mayúsculas. Por ejemplo:

```
>>> PI = 3.1416

>>> G = 9.8
```

1.9. Operación de asignación

La acción de asociar un valor a una variable se denomina “asignación”. En Python, la asignación establece el tipo de datos que se van a representar y asocia la variable con el dato. En Python basta con asignar un valor a una variable para que se cree la variable con un tipo acorde al valor asignado.

Podemos distinguir entre asignación interna y asignación externa. La asignación interna da un valor a una variable desde el programa. La externa lo hace desde un dispositivo de entrada de datos: un teclado, ratón, etc. En este apartado se va a ver la primera, dejando para otra unidad la operación de asignación externa.

1.9.1. Asignación interna

El operador de asignación interna en Python es “=”. Su sintaxis es:

```
variable = <expresión>
```

En donde variable es el identificador correspondiente a una variable que se use en el programa.

Por ejemplo, si se asigna:

```
>>> n1=12;

>>> mayor=”Soy mayor de edad”

>>> pi=3.1416
```

Se crean automáticamente las variables n1, mayor y pi, cada una con un tipo acorde al valor que se les ha asignado. Así, n1 será entero, mayor será una cadena de caracteres y pi un número real en coma flotante. Se dice que se trata de una asignación dinámica de tipos, frente a la asignación estática típica de C o Java. La asignación estática de tipos es preferible en el caso de programas compilados ya que permite optimizar la operación de compilación. La marca >>> simplemente indica que las sentencias se escriben a continuación de ella como se verá en esta misma unidad posteriormente.

La semántica de esta operación es:

- En primer lugar se determina el tipo correspondiente a la expresión que figura en el segundo miembro, al objeto de reservar en memoria el espacio necesario para almacenar el dato.
- En segundo lugar se evalúa expresión.
- Se asigna dicho valor a variable.

La operación de asignación múltiple permite asignar valores usando un solo símbolo de asignación, separando por comas las variables y las expresiones correspondientes a cada variable. Por ejemplo:

```
>>> x, y, z = 0, 0.5, "A"
```

Asigna a x el valor entero 0, a y el valor real 0.5 y a z el carácter A.

Expresión es una combinación de operandos y operadores, escrito con la sintaxis adecuada a los operadores empleados, que representa un determinado cálculo a realizar.

Evaluación de las expresiones.

Cuando en una expresión hay varios operadores involucrados, es necesario saber el orden en que se opera con ellos. El protocolo de evaluación de una expresión asigna mayor prioridad a unos operadores frente a otros, tal como se muestra en la siguiente tabla para los operadores aritméticos.

Mayor prioridad	Exponente : **
	Negación : -
	Multiplicación, división, división entera, módulo : *, /, //, %
Menor prioridad	Suma, resta: +, -

Sin embargo, se pueden emplear paréntesis para forzar el orden en que se quieren realizar las operaciones parciales. En primer lugar se evalúan los paréntesis más interiores.

En caso de presentarse un conflicto entre operadores de un mismo nivel de prioridad, la expresión se evalúa de izquierda a derecha.

Python permite una forma abreviada de asignación, como la que se representa en la siguiente tabla:

Operador	Ejemplo	Significado
=	x = x+y	Asignación típica.
+=	x += y	x = x+y
-=	x -= y	x=x-y
/=	x /= y	x=x/y
*=	x *= y	x=x*y
**=	x**=y	x elevado a y
//=	x//=y	x=x//y devuelve el cociente entero de dividir x entre y
%=	x %= y	x=x%y devuelve el resto de dividir x entre y

1.10. Composición secuencial de instrucciones

Una computación es una secuencia de acciones que transforma un estado inicial en otro final. Supondremos que la instrucción, también llamada sentencia, es la unidad básica de acción.

EJEMPLO

Intercambio de valor entre dos variables. Consideremos dos variables x e y a las que se asignan valores enteros. Si inicialmente x toma el valor X e y toma el valor Y, intercambiar sus valores implica que tras efectuarse la composición de intercambio, la variable x toma el valor Y, y la variable y toma el valor X. Sin embargo, para hacer esto necesitamos una tercera variable a la que podemos llamar z. Hacemos:

```
>>> x, y = 10, 3
>>> z=x  #  z pasa a valer 10, x e y no cambian
de valor: x=10, y=3
>>> x=y  #  x pasa a valer 3, z e y no cambian:
z=10, y=10
>>> y=z  #  y pasa a valer 10, x y z no cambian:
x=3, z=10
```

Almacenando el valor de la variable x en la variable z. Como hemos guardado en z su valor, ya lo podemos cambiar haciendo que x pase a valor lo que vale y. Finalmente damos a y el valor de z.

En el desarrollo anterior se aprecia cómo va cambiando el estado de las variables x, y, z hasta llegar al objetivo buscado. La variable auxiliar z es un ejemplo de una clase de variables que se emplean para almacenar información de manera temporal. Se denominan “variables locales” de un algoritmo.

En Python esto se podría hacer de una forma muy sencilla y elegante utilizando la asignación múltiple:

```
>>> x, y = y, x
```

EJEMPLO

Ejemplo de secuencia: conversión de una cantidad positiva de segundos a su equivalente en horas, minutos y segundos, que se satisface con la siguiente secuencia:

```
>>> seg=58000
>>> min= seg // 60          # division entera
>>> horas= min //60         # division entera
>>> seg=seg % 3600          # resto
>>> print (“horas: “, horas, “ minutos: “, min, “
segundos: “, seg)
```

1.11. La salida estandar en Python

Cuando se escribe un programa en Python, la escritura del texto en la pantalla se realiza mediante la función *print()*. Esta función toma como argumento unos parámetros separados por comas y los escribe en la salida estándar. Los parámetros pueden ser un número, una cadena de caracteres (es una sucesión de caracteres puestos entre comillas) u otro objeto, pero antes de escribirlos los transforma en cadena. Cuando hay varios parámetros al escribirlos van separados por un espacio en blanco. La forma de escribir estas funciones es:

```
>>> print (lista_de_parámetros)
```

EJEMPLO

Veamos un ejemplo:

```
>>> x=("coche", "camión", "moto")
>>> print(x)
('coche' 'camión' 'moto')
```

Se pueden incluir separadores entre dos cadenas de caracteres mediante un parámetro:

```
sep=<separador>
```

en donde <separador> es una cadena de caracteres usada como separador.

Por ejemplo:

```
>>> print("Hola", "Mundo", sep="___")
Hola___Mundo
```

El separador también se puede usar para evitar que escriba el espacio en blanco entre dos parámetros. Por ejemplo:

```
>>> print("a", "b", sep="")
ab
```

o se puede aplicar en múltiples ocasiones cuando le anteceden más de dos parámetros:

```
>>> print(123, 456, 789, sep=".")
123.456.789
```

en donde se ve que el separador se aplica entre cada elemento de la lista de parámetros.

El operador "+" concatena (une) dos cadenas de caracteres. Por ejemplo, si se escribe:

```
>>> print( "El carácter ("  +  'a' +  ") tiene el valor "  +  "97"  );
```

imprime en la pantalla:

El carácter (a) tiene el valor 97

1.12. Comentarios

En cualquier punto de un programa se pueden escribir comentarios cuya misión es aclarar algunas cuestiones a la persona que lee el programa. Estos comentarios ayudan posteriormente al mantenimiento del programa, ya que cuando pasa cierto tiempo, el programador se olvida de detalles acerca de cómo hizo el programa. El compilador hace caso omiso de los comentarios y no surten ningún efecto en el código.

Una línea de comentarios comienza por el carácter #. A partir de ese carácter y hasta el fin de línea lo ignara el intérprete de Python.

EJEMPLO

Por ejemplo:

```
# Esto es un comentario.
```

1.13. Palabras reservadas en Python

En Python hay palabras reservadas que no se pueden usar para dar nombres de variables o funciones. A continuación se citan, aunque no es necesario que el alumno las aprenda ya que lo irá haciendo conforme avance el curso. Son:

nad	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	not	or	pass
print	raise	return	try	while	with
yield					

1.14. Modos de trabajo en Python

Hay dos formas de utilizar Python:

- Modo interactivo, como si fuese una calculadora. Conforme se escribe una expresión o una sentencia se ejecuta con sólo pulsar la tecla Enter.
- Modo "*script*". Se escribe un fichero que contenga el código del programa que se quiere realizar. Basta con guardar el fichero con una extensión ".py". Tras eso, se puede ejecutar el fichero.

RESUMEN

- Un ordenador es una máquina programable para el tratamiento de la información. Sus aspectos esenciales son, que es programable (ejecuta programas) y que trata con información en forma de datos.
- Los componentes básicos de un ordenador son: unidad central de proceso, memoria y periféricos.
- El sistema operativo es un programa de gestión y control de la máquina que hace que el usuario y el programador se desentiendan de aspectos relacionados con su arquitectura y su funcionamiento interno.
- Programa fuente es el programa escrito en un lenguaje de alto nivel como Python.
- Programa objeto es un programa que la máquina sabe ejecutar. Para ello, se necesita un programa traductor que traduce el programa fuente en el programa objeto.
- El programa de *bytecodes* es un programa objeto para una máquina virtual que es independiente de la máquina real y del sistema operativo usado.
- El fichero que contiene el programa en código fuente de Python tiene por extensión "*py*". El fichero compilado de *bytecodes* por extensión "*.pyc*".
- Un dígito es un número del 0 al 9.
- Un identificador es un nombre que se da a algo.
- Un identificador debe empezar por una letra seguida de letras o dígitos o el carácter de subrayado.
- Las letras mayúsculas y las minúsculas son distintas en los identificadores.
- Un literal booleano puede tomar los valores *True* o *False*.
- Una línea de comentarios comienza por el carácter #.
- La operación de asignación interna usa el símbolo =.
- La operación de escritura desde el programa usa la sentencia *print*().