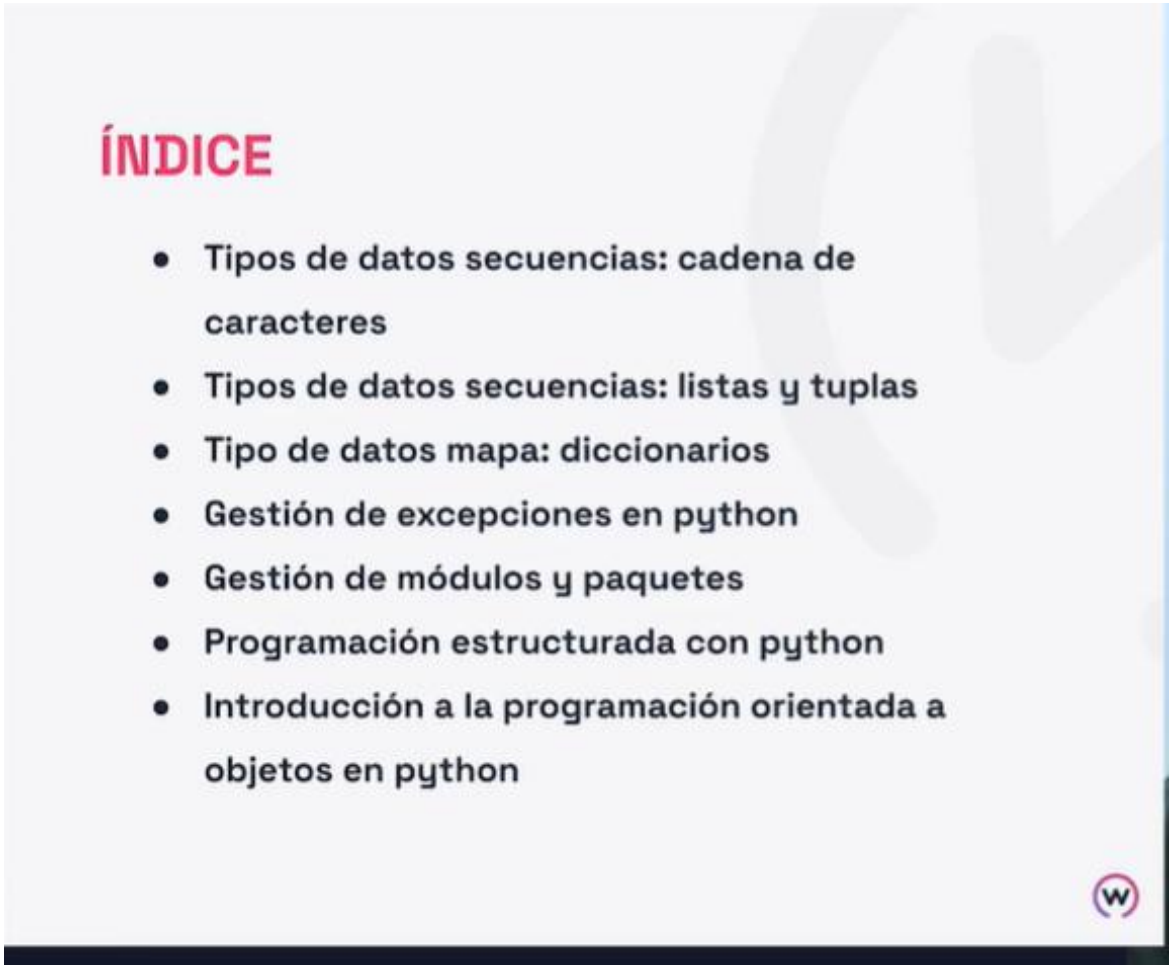
Python



Un **programador** es una persona que resuelve problemas, y para llegar a ser un programador eficaz se **necesita aprender a resolver problemas de un modo riguroso y sistemático**:

- *Definición o análisis del problema*: consiste en el estudio detallado del problema. Se debe identificar los datos de entrada, de salida y la descripción del problema.
- *Diseño del algoritmo*: que describe la secuencia ordenada de pasos que conduce a la solución de un problema dado: **algoritmo**.
- *Transformación del algoritmo en un programa (codificación)*: Se expresa el algoritmo como un programa en un **lenguaje de programación**.
- *Ejecución y validación del programa*.

Análisis del problema

El primer paso, análisis del problema, requiere un estudio a fondo del problema y de todo lo que hace falta para poder abordarlo.

El propósito del análisis de un problema es ayudar al programador (Analista) para llegar a una cierta comprensión de la naturaleza del problema. Una buena definición del problema, junto con una descripción detallada de las especificaciones de entrada/salida, son los requisitos más importantes para llegar a una solución eficaz.

Para realizar el análisis del problema realizamos varios pasos:

- **Definición del problema**: Tenemos que entender y comprender la naturaleza del problema, tenemos que dominar los conceptos con los que estamos trabajando y conocer que operaciones hay que realizar para solucionar el problema y obtener la información de salida.
- **Especificación de los datos de entrada**: Hay que determinar que datos de entrada necesitamos para resolver el problema.
- **Especificación de los datos de salida**: También hay que determinar los datos de salida que van a ofrecer la solución del problema.

Diseño de algoritmos

A partir de los requerimientos, resultados del análisis, empieza la etapa de **diseño** donde tenemos que construir un **algoritmo** que resuelva el problema.

Definición de algoritmo

Un **algoritmo** es un conjunto de acciones que especifican la secuencia de operaciones realizar, en orden, para resolver un problema.

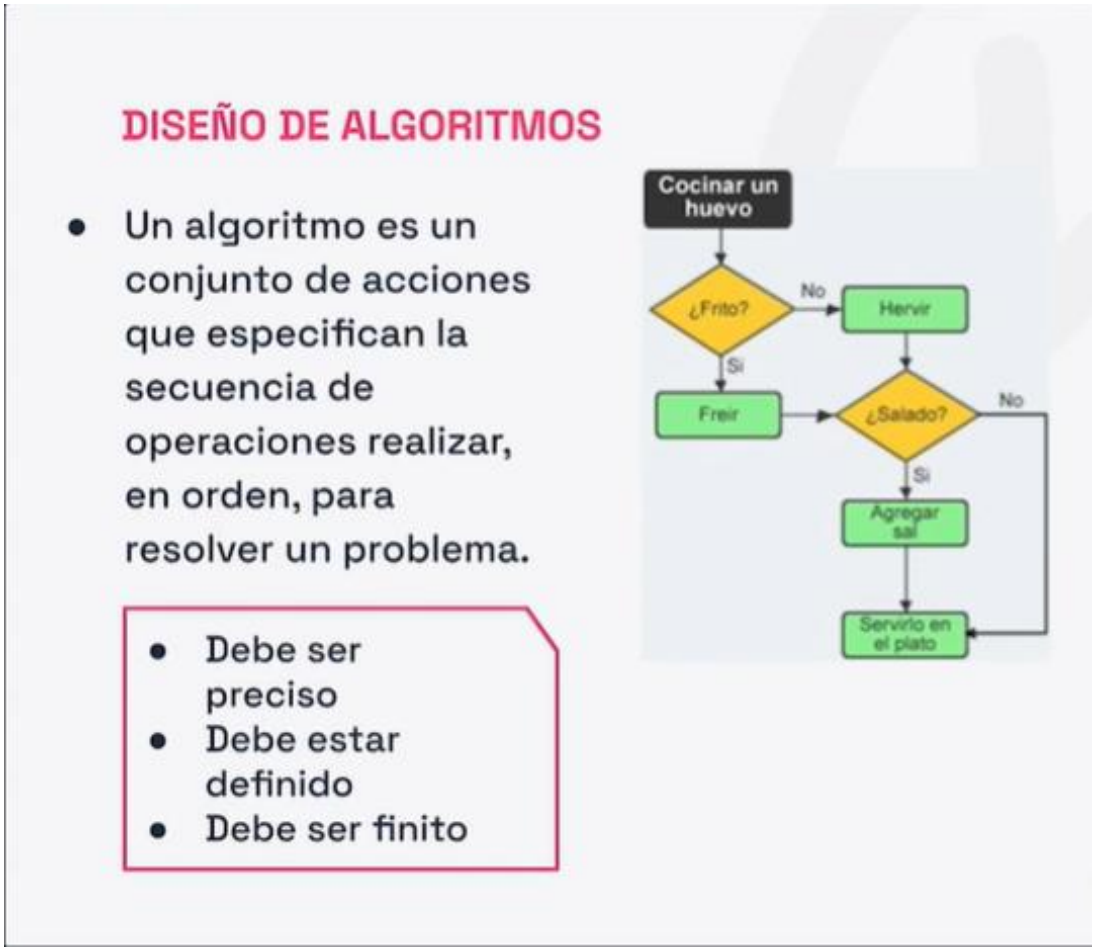
Los algoritmos son independientes tanto del lenguaje de programación como del ordenador que los ejecuta.

Las características de los algoritmos son:

- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.

Herramientas de representación de algoritmos

- Un **diagrama de flujo** es una de las técnicas de representación gráfica de algoritmos más antiguas. Ventajas: permite altos niveles de estructuración y modularización y es fácil de usar. Desventajas: son difíciles de actualizar y se complican cuando el algoritmo es grande.
- El **pseudocódigo**, nos permite una aproximación del algoritmo al lenguaje natural y por tanto un a redacción rápida del mismo.



Contenido de la lección

Para que un ordenador realice un proceso, se le debe suministrar en primer lugar un algoritmo adecuado, que llamamos **programa**. El procesador debe ser capaz de interpretar el algoritmo, lo que significa:

- Comprender las instrucciones de cada paso.
- Realizar las operaciones correspondientes.

Clasificación de los lenguajes de programación

Lenguaje Máquina

Son aquellos que están escritos en lenguajes que directamente entiende el ordenador, ya que sus instrucciones son cadenas binarias (secuencias de ceros y unos) que especifican una operación y las posiciones (dirección) de memoria implicadas en la operación. Se denominan instrucciones de máquina o **código máquina**. Características:

- Las instrucciones en lenguaje máquina dependen del hardware del ordenador y por tanto serán diferentes de un ordenador a otro.
- Se puede transferir un programa a memoria sin necesidad de traducción posterior, lo que supone una mayor velocidad de ejecución a cualquier otro lenguaje.
- Dificultad y lentitud en la codificación.

- Conjunto de instrucciones reducido (operaciones muy elementales)

Lenguaje de bajo nivel (Ensamblador)

Los lenguajes de bajo nivel son más fáciles de utilizar que los lenguajes máquina, pero, al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el **ensamblador**.

Las instrucciones en lenguaje ensamblador son instrucciones conocidas como mnemónicos.

Un programa escrito en lenguaje ensamblador no puede ser ejecutado directamente por la máquina, sino que requiere una fase de traducción al lenguaje máquina. El programa original escrito en lenguaje ensamblador se denomina **programa fuente** y el programa traducido en lenguaje máquina se conoce como **programa objeto**, ya directamente entendible por el ordenador.

Lenguaje de alto nivel

Los lenguajes de alto nivel son los mas utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Otra razón es que un programa escrito en un lenguaje de alto nivel es independiente de la máquina, o sea, las instrucciones del programa del ordenador no dependen del diseño hardware de un ordenador en particular. Por lo tanto los programas escritos en lenguajes de alto nivel son portables o transportables, lo que significa la posibilidad de poder ser ejecutados con poca o ninguna modificación en diferentes tipos de ordenadores.

Al igual que sucede con los lenguajes ensambladores, los programas fuente tienen que ser traducidos por **programas traductores**, llamados en este caso compiladores o intérpretes.

Ejemplos de lenguajes de programación de alto nivel:

BASIC, COBOL, PASCAL, C, VISUAL BASIC, JAVA, PYTHON, PERL, GO, PHP, RUBY,…

Programas traductores

Los traductores transforman programas escritos en un lenguaje de alto nivel en programas escritos en código máquina. Podemos indicar distintos tipos:

Compiladores

- Convierte un programa escrito en alto nivel (código fuente) a un programa máquina (código ejecutable).
- Para generar el código ejecutable el código no debe tener errores de sintaxis.
- Necesitamos un compilador para cada arquitectura y sistema operativo.
- Los programas ejecutables no son compatibles entre plataformas.
- Una vez generado el programa ejecutable, no es necesario tener el código fuente.
- Ejemplos: C, Pascal,…

Interpretes

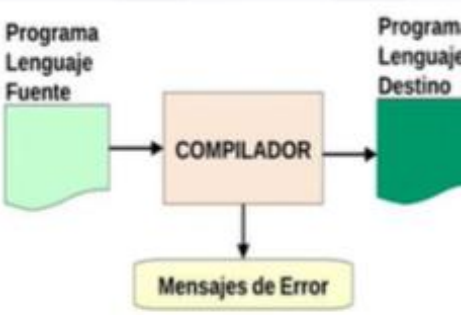
- La traducción y ejecución de código fuente a código máquina se hace línea por línea.
- Los errores de sintaxis aparecen cuando se interpreta la instrucción con error.
- Necesitamos el código fuente para ejecutar el programa.
- Los lenguajes interpretados suelen ser más lentos en su ejecución
- Ejemplos: Python, PHP, …

Máquina virtual

- La traducción se hace en dos pasos.
- Primero se compila el código fuente a un código intermedio (bytecode).
- Segundo, este bytecode se interpreta y ejecuta por una “máquina virtual”.
- El bytecode es multiplataforma.
- Necesito una “máquina virtual” para cada plataforma.
- No necesito el código fuente.
- Ejemplo: Java, C#, …

COMPILADOR

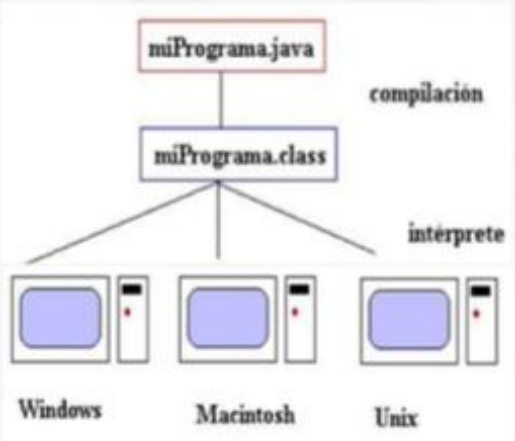
- Convierte un programa escrito en alto nivel (código fuente) a un programa máquina (código ejecutable)
- Para generar código ejecutable el código no debe tener errores de sintaxis
- Necesitamos un compilador para cada arquitectura y sistema operativo
- Los programas ejecutables no son compatibles entre plataformas
- Una vez generado el programa ejecutable, no es necesario tener el código fuente.



Ejemplos:
C, Pascal, ...

MÁQUINA VIRTUAL

- La traducción se hace en dos pasos
- Primero se compila el código fuente a un código intermedio (bytecode)
- Segundo, este bytecode se interpreta y ejecuta por una "máquina virtual"
- El bytecode es multiplataforma
- Necesito una MV para cada plataforma
- No necesito el código fuente



ia.

Ejemplos: java, c#, ...

Características de Python

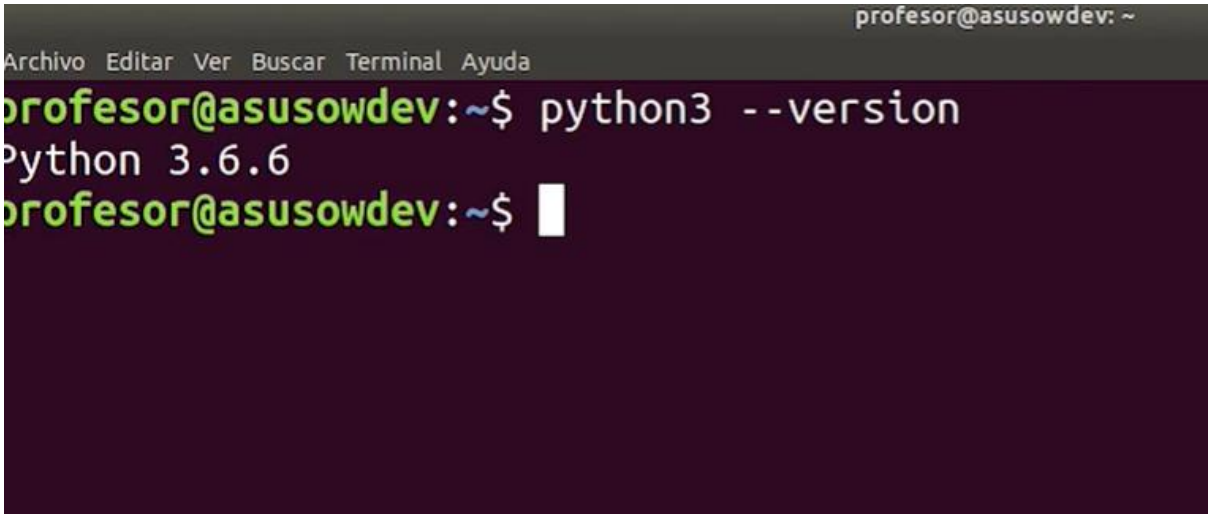
Python es un lenguaje:

- Interpretado
- Alto nivel
- Multiparadigma, ya que soporta **orientación a objetos**, **programación imperativa** y **programación funcional**.
- Multiplataforma
- Libre

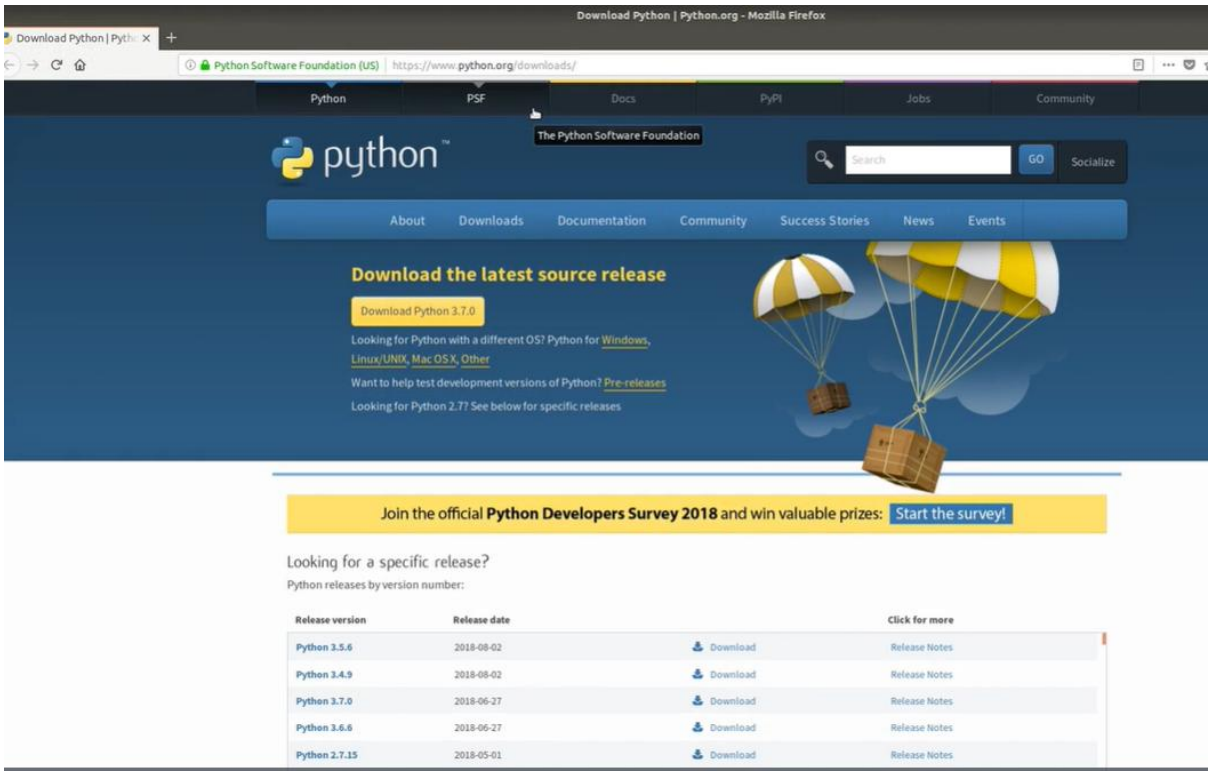
¿Por qué elegir python?

- Porque es fácil de aprender
- Sintaxis muy limpia y sencilla
- Hay que escribir menos
- Obtienes resultados muy rápido
- Puedes programar con distintos paradigmas:
- Programación imperativa
- Orientación a objetos
- Programación funcional
- Puedes programas distintos tipos de aplicaciones:
- Aplicaciones de escritorio
- Aplicaciones web
- Scripts
- Muchos usan Python (Google, Nokia, IBM). Es demandado.
- Gran cantidad de módulos, muchísimas funcionalidades.
- Una gran comunidad que apoya el proyecto.
- Viene preinstalado en la mayoría de sistemas

Para linux, viene instalado por defecto



Para window



3.1. Installing Python

Unlike most Unix systems and services, Windows does not include a system supported installation of Python. To make Python available, the CPython team has compiled Windows installers (MSI packages) with every release for many years. These installers are primarily intended to add a per-user installation of Python, with the core interpreter and library being used by a single user. The installer is also able to install for all users of a single machine, and a separate ZIP file is available for application-local distributions.

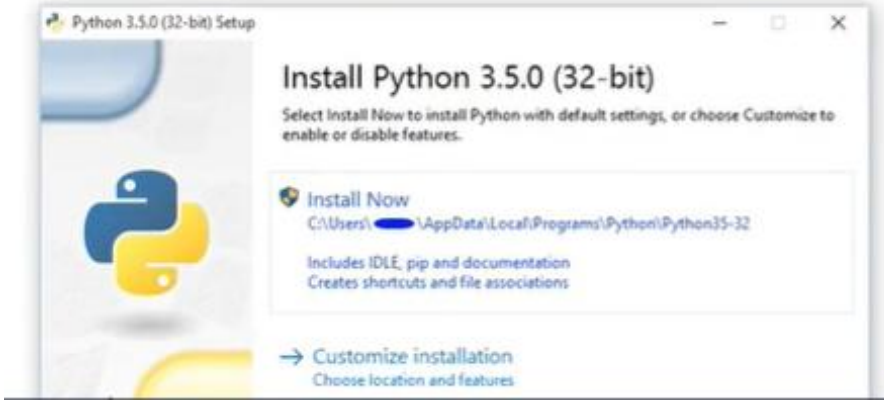
3.1.1. Supported Versions

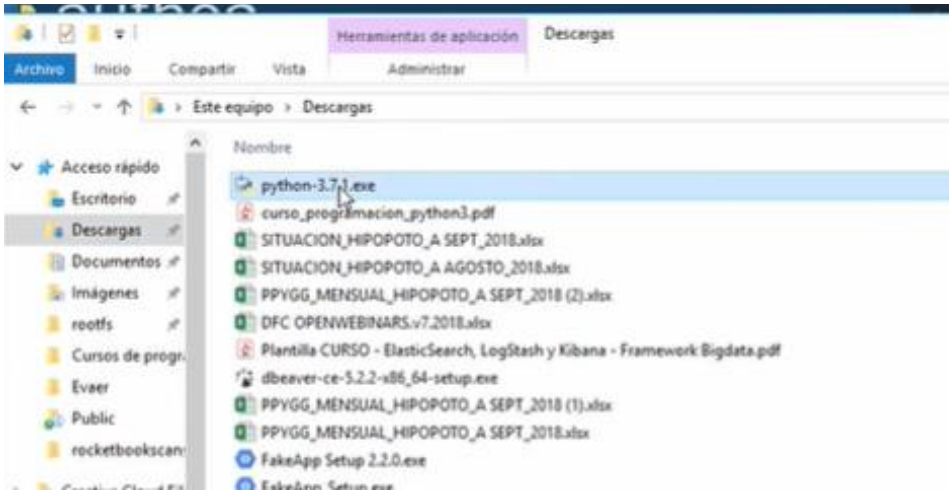
As specified in **PEP 11**, a Python release only supports a Windows platform while Microsoft considers the platform under extended support. This means that Python 3.7 supports Windows Vista and newer. If you require Windows XP support then please install Python 3.4.

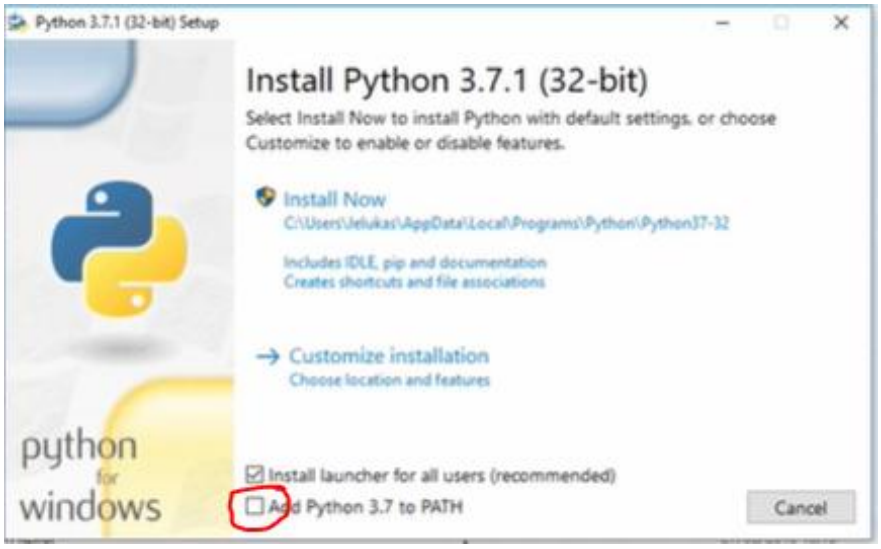
3.1.2. Installation Steps

Four Python 3.7 installers are available for download - two each for the 32-bit and 64-bit versions of the interpreter. The *web installer* is a small initial download, and it will automatically download the required components as necessary. The *offline installer* includes the components necessary for a default installation and only requires an internet connection for optional features. See Installing Without Downloading for other ways to avoid downloading during installation.

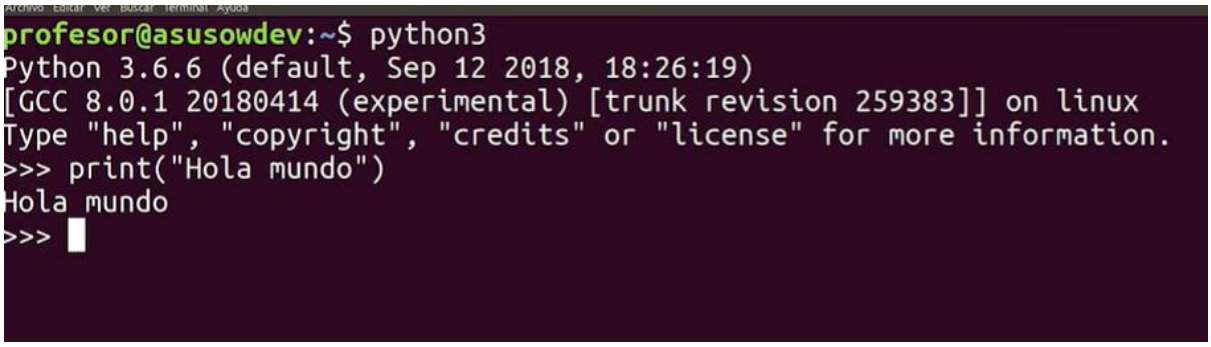
After starting the installer, one of two options may be selected:







El intérprete nos puede servir para ejecutar pruebas, para ver como funciona.



Uso del interprete

Al instalar python3 el ejecutable del interprete lo podemos encontrar en /usr/bin/python3 (en debian Stretch). Este directorio por defecto está en el PATH, por lo tanto lo podemos ejecutar directamente en el terminal, para ello ejecutamos:

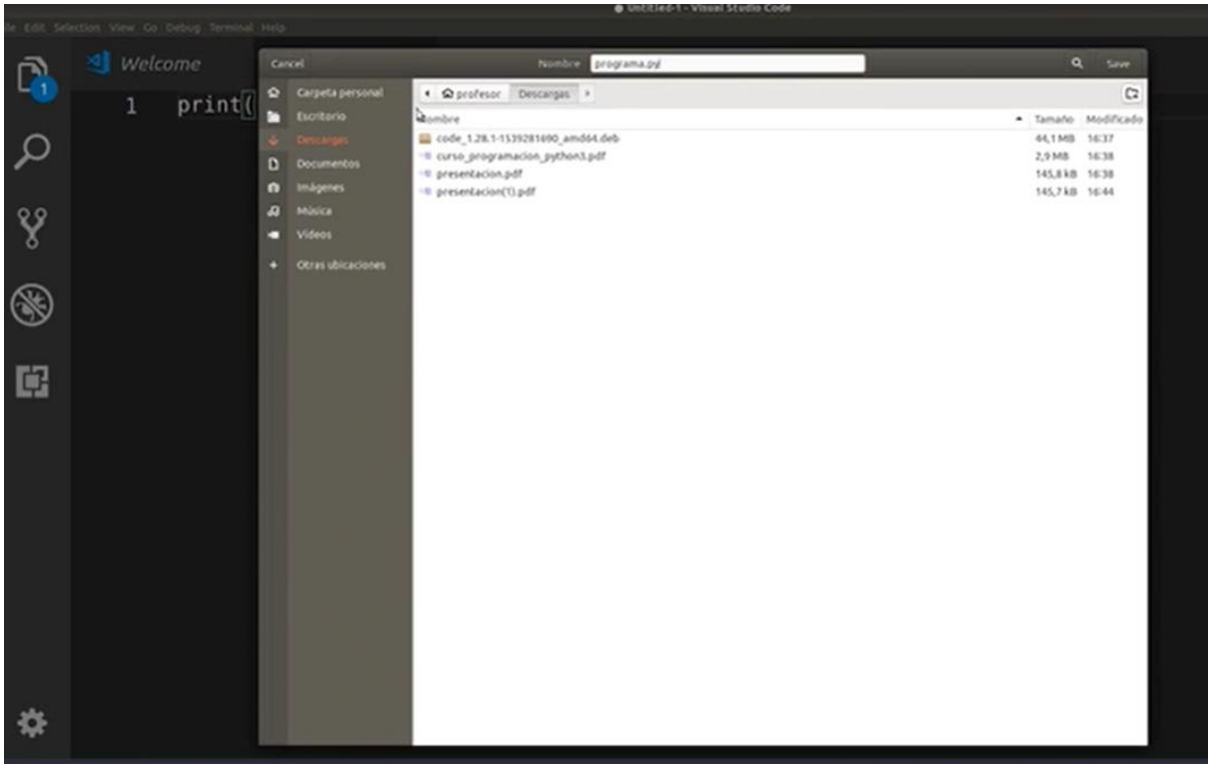
$ python3 Python 3.5.3 (default, Jan 19 2017, 14:11:04) [GCC 6.3.0 20170118] on linux Type "help", "copyright", "credits" or "license" for more information. >>>
A partir de un fichero con el código fuente

Si tenemos nuestro programa en un fichero fuente (suele tener extensión py), por ejemplo programa.py,lo ejecutaríamos de la siguiente manera.

$ python3 programa.py
Para escribir un fichero con el código fuente de nuestro programa tenemos varias opciones:

- Podemos usar un IDE (entorno de desarrollo integrado), que además de la posibilidad de editar el código, nos ofrezca otras herramientas: depuración de código, generación automático de código, ayuda integrada, manejo del proyecto, gestión de los sistemas de control de versiones,..). Existen muchos IDE a nuestra disposición: Entornos de desarrollo para python.
- En este curso vamos usar un editor de texto: Nos permiten escribir nuestro código fuente de manera eficiente, además los nuevos editores de texto permiten añadir funcionalidades a

través de plugin, algunos ejemplos: Sublime Text, Atom, Visual Studio Code, vim, emacs,… aunque hay muchos más: Editores de texto para python.




```
profesor@asusowdev:~$ python3
Python 3.6.6 (default, Sep 12 2018, 18:26:19)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hola mundo")
Hola mundo
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>>
profesor@asusowdev:~$ cd Descargas/
profesor@asusowdev:~/Descargas$ ls
 code_1.28.1-1539281690_amd64.deb  'presentacion(1).pdf'   programa.py
 curso_programacion_python3.pdf     presentacion.pdf
profesor@asusowdev:~/Descargas$ python3 programa.py
Hola Mundo
profesor@asusowdev:~/Descargas$
```

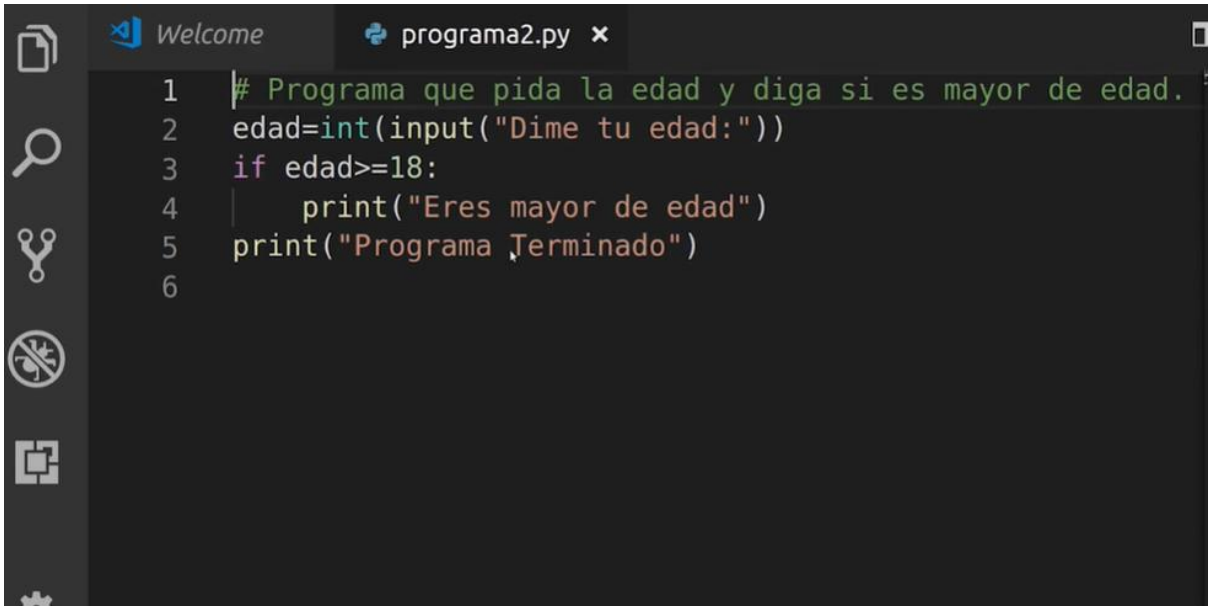
MI PRIMER PROGRAMA EN PYTHON 3

Pseudocódigo

```
Proceso mayor_edad
    Definir edad como entero; Escribir "Dime tu edad:"; Leer edad;
    Si edad>=18 Entonces
        Escribir "Eres mayor de edad"; FinSi
    Escribir "Programa terminado"; FinProceso
```

Python 3

```
# Programa que pida la edad
# y diga si es mayor de edad.
edad=int(input("Dime tu edad:"))
if edad>=18:
print("Eres mayor de edad")
print("Programa Terminado")
```




```
profesor@asusowdev:~/Descargas$ python3 programa2.py
Dime tu edad:20
Eres mayor de edad
Programa Terminado
profesor@asusowdev:~/Descargas$
```

LITERALES, VARIABLES Y EXPRESIONES

- Literales enteros: 3, 12, -23
- Literales reales: 12.3, 45.6
- Literales cadenas:
  'hola que tal!'
  "Muy bien"
  '''Podemos \n ir al cine'''

Variables:

```
>>> var = 5
>>>
var  5
```

Expresiones

```
a + 7
(a ** 2) + b
```

OPERADORES

- Operadores aritméticos: +, -, *, /, //, %, **.
- Operadores de cadenas: +, *
- Operadores de asignación: =
- Operadores de comparación: ==, !=, >=, >, <=, <
- Operadores lógicos: and, or, not
- Operadores de pertenencia: in, not in

PRECEDENCIA DE OPERADORES

1. Los paréntesis rompen la precedencia.
2. La potencia (**)
3. Operadores unarios (+ -)
4. Multiplicar, dividir, módulo y división entera (* % //)
5. Suma y resta (+ -)
6. Operador binario AND (&)
7. Operadores binario OR y XOR (^ |)
8. Operadores de comparación (<= < > >=)
9. Operadores de igualdad (<> == !=)
10. Operadores de asignación (=)
11. Operadores de pertenencia (in, in not)
12. Operadores lógicos (not, or, and)

TIPOS DE DATOS

- Tipos numéricos
  - Tipo entero (int)
  - Tipo real (float)
- Tipos booleanos (bool)
- Tipo de datos secuencia
  - Tipo lista (list)
  - Tipo tuplas (tuple)
- Tipo de datos cadenas de caracteres
  - Tipo cadena (str)
- Tipo de datos mapas o diccionario (dict)

```
>>> type(5)
<class 'int'>
>>> type(5.5)
<class 'float'>
>>> type([1,2])
<class 'list'>
>>> type(int)
<class 'type'>
```

Una tupla en Python es una
colección ordenada e inmutable de elementos, lo que significa que una vez creada, sus elementos no pueden ser modificados, añadidos ni eliminados. Se definen usando paréntesis () y se diferencian de las listas en que no se pueden alterar sus contenidos, haciéndolas ideales para guardar datos que no deben cambiar.

Características principales

- **Inmutabilidad:** La diferencia clave con las listas es que las tuplas no se pueden modificar después de su creación.
- **Orden:** Los elementos tienen un orden fijo y se puede acceder a ellos a través de su índice (empezando desde

  0 0
  0
).
- **Heterogeneidad:** Pueden almacenar elementos de diferentes tipos de datos (números, cadenas, etc.).
- **Sintaxis:** Se definen con paréntesis (). Por ejemplo: mi_tupla = (1, 'hola', 3.14).

¿Cuándo usar una tupla?

- **Datos fijos:** Son perfectas para datos que deben permanecer constantes, como las coordenadas (X, Y) o la información de una fecha.
- **Seguridad:** Al ser inmutables, protegen los datos de cambios accidentales, lo que mejora la seguridad del código.
- **Eficiencia:** En general, las tuplas son más rápidas que las listas, lo que puede ser una ventaja en ciertas operaciones.

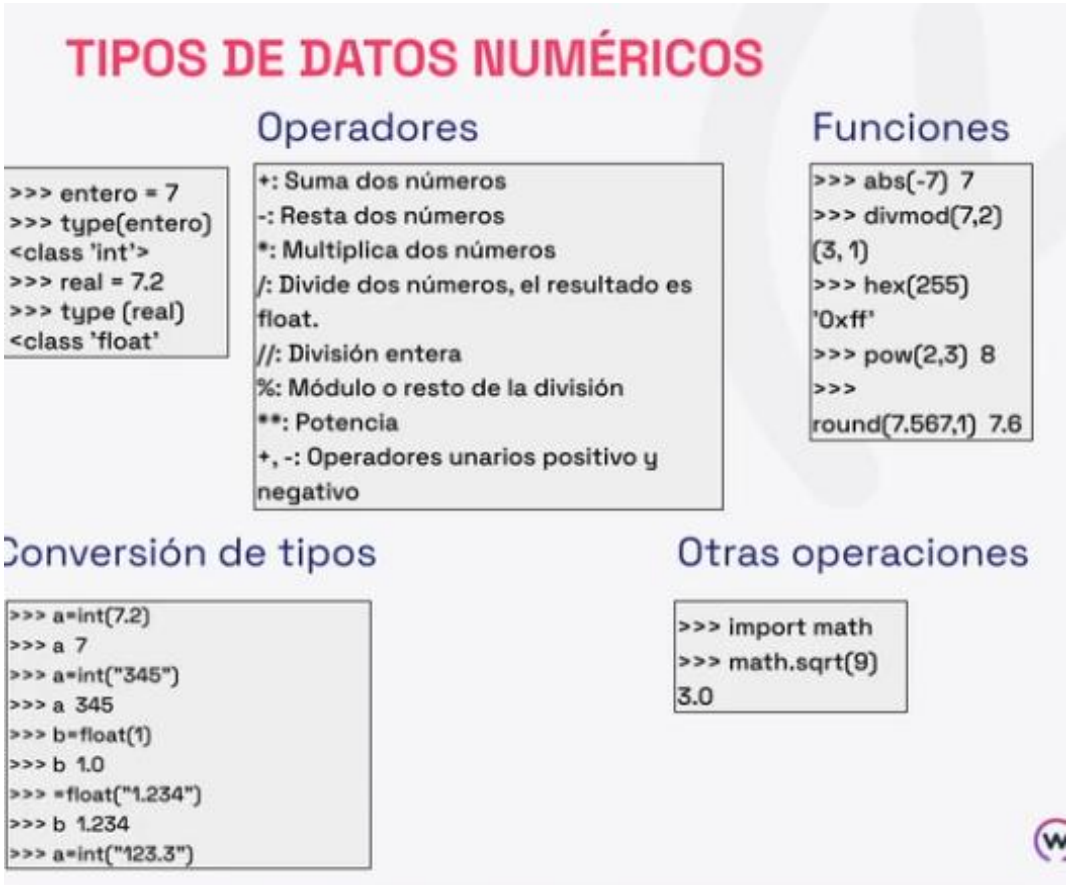
```
>>> type(var)
<class 'str'>
>>> var = True
>>> type(var)
<class 'bool'>
```

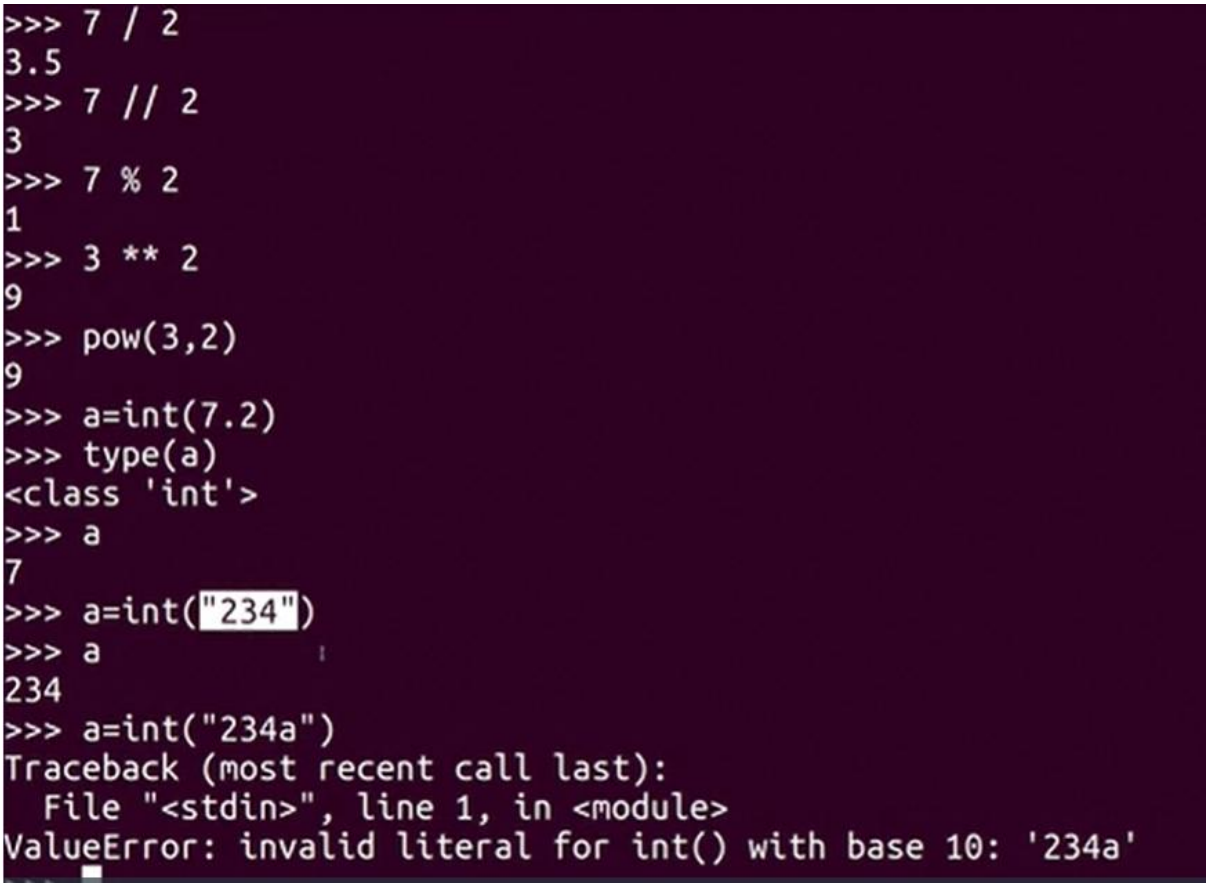
Para definir los tipos booleanos se indica True o False, con el primer carácter en mayúsculas.

Python3 trabaja con varios tipos numéricos, pero en este curso sólo vamos a utilizar dos:

- **Enteros (int):** Representan todos los números enteros (positivos, negativos y 0), sin parte decimal. En python3 este tipo no tiene limitación de espacio.
- **Reales (float)**: Sirve para representar los números reales, tienen una parte decimal y otra decimal. Normalmente se utiliza para su implementación un tipo double de C.
  Ejemplos

>>> entero = 7 >>> type(entero) <class 'int'> >>> real = 7.2 >>> type (real) <class 'float'





Python3 trabaja con varios tipos numéricos, pero en este curso sólo vamos a utilizar dos:

- Enteros (int): Representan todos los números enteros (positivos, negativos y 0), sin parte decimal. En python3 este tipo no tiene limitación de espacio.
- Reales (float): Sirve para representar los números reales, tienen una parte decimal y otra decimal. Normalmente se utiliza para su implementación un tipo double de C.
  Ejemplos

  >>> entero = 7 >>> type(entero) <class 'int'> >>> real = 7.2 >>> type (real) <class 'float'

Operadores aritméticos

- +: Suma dos números
- -: Resta dos números
- *: Multiplica dos números
- /: Divide dos números, el resultado es float.
- //: División entera
- %: Módulo o resto de la división
- **: Potencia
- +, -: Operadores unarios positivo y negativo
  Funciones predefinidas que trabajan con números:

- abs(x): Devuelve al valor absoluto de un número.
- divmod(x,y): Toma como parámetro dos números, y devuelve una tubla con dos valores, la división entera, y el módulo o resto de la división.
- hex(x): Devuelve una cadena con la representación hexadecimal del número que recibe como parámetro.

- bin(x): Devuelve una cadena con la representación binaria del número que recibe como parámetro.
- pow(x,y): Devuelve la potencia de la base x elevado al exponente y. Es similar al operador**`.
- round(x,[y]): Devuelve un número real (float) que es el redondeo del número recibido como parámetro, podemos indicar un parámetro opcional que indica el número de decimales en el redondeo.
  Ejemplos

  >>> abs(-7) 7 >>> divmod(7,2) (3, 1) >>> hex(255) '0xff' >>> pow(2,3) 8 >>> round(7.567,1) 7.6
  Conversión de tipos

- int(x): Convierte el valor a entero.
- float(x): Convierte el valor a float.
  Los valores que se reciben también pueden ser cadenas de caracteres (str).

  Ejemplos

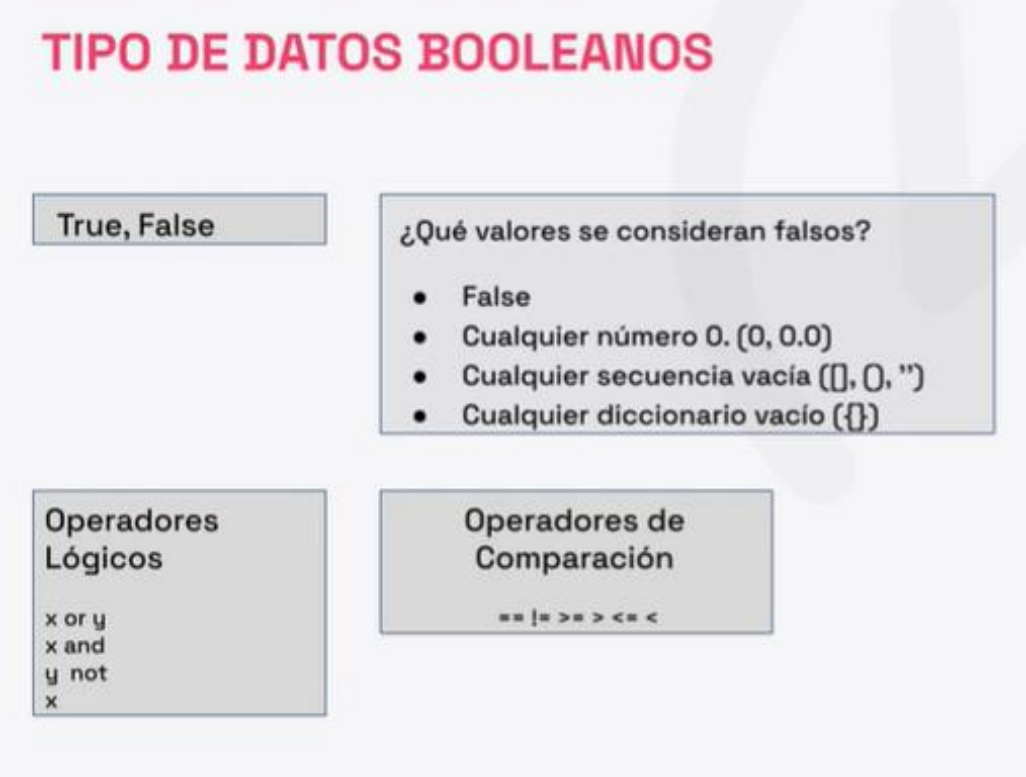
  >>> a=int(7.2) >>> a 7 >>> type(a) <class 'int'> >>> a=int("345") >>> a 345 >>> type(a) <class 'int'> >>> b=float(1) >>> b 1.0 >>> type(b) <class 'float'> >>> b=float("1.234") >>> b 1.234 >>> type(b) <class 'float'>
  Por último si queremos convertir una cadena a entero, la cadena debe estar formada por caracteres numéricos, sino es así, obtenemos un error:

  a=int("123.3") Traceback (most recent call last): File "<stdin>", line 1, in <module> ValueError: invalid literal for int() with base 10: '123.3'

  Otras operaciones del modulo de matemáticas, primero hago un import

  Quizás eches en falta más operaciones que podemos realizar sobre los números. En el **módulo math** encontramos muchas de estas operaciones. Para utilizarlas vamos a importar el módulo, por ejemplo para realizar una raíz cuadrada:

  >>> import math >>> math.sqrt(9) 3.0



Tipo booleano

El tipo booleano o lógico se considera en python3 como un subtipo del tipo entero. Se puede representar dos valores: verdadero o false (True, False).

¿Qué valores se interpretan como FALSO?

Cuando se evalúa una expresión, hay determinados valores que se interpretan como False:

- False
- Cualquier número 0. (0, 0.0)
- Cualquier secuencia vacía ([], (), ‘’)
- Cualquier diccionario vacío ({})

Operadores de comparación

Las expresiones lógicas utilizan operadores de comparación, me permiten comparar dos valores y devuelven un valor booleano, dependiendo de lo que este comparando.

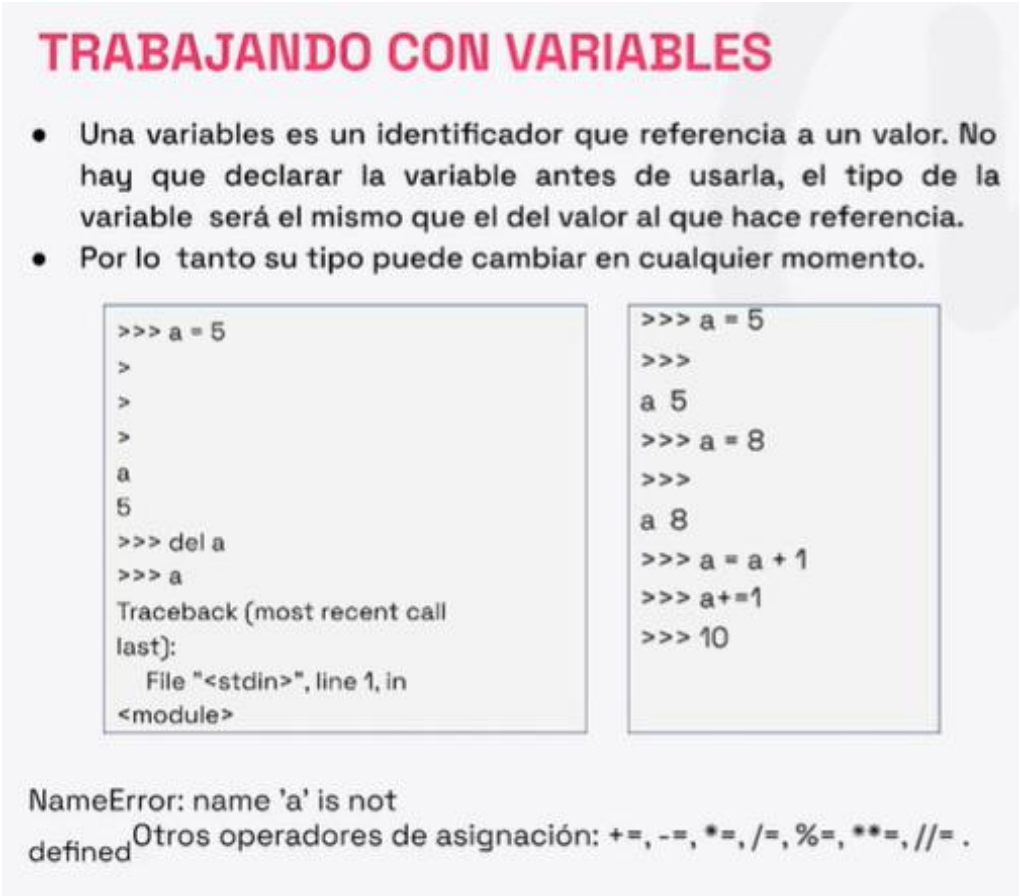
- ==: Igual que
- !=: Distinto que
- >: Mayor que
- <: Menor que
- <=: Menor o igual
- >=: Mayor o igual

Operadores booleanos o lógicos

Los operadores booleanos se utilizan para operar sobre expresiones booleanas y se suelen utilizar en las estructuras de control alternativas (if, while):

- x or y: Si x es falso entonces y, sino x. Este operados sólo evalúa el segundo argumento si el primero es False.
- x and y: Si x es falso entonces x, sino y. Este operados sólo evalúa el segundo argumento si el primero es True.
- not x: Si x es falso entonces True, sino False.

```
profesor@asusowdev:~/Descargas$ python3
Python 3.6.6 (default, Sep 12 2018, 18:26:19)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> num = 7
>>> num == 7
True
>>> indicador = True
>>> type(indicador)
<class 'bool'>
>>> num >= 1 and num<=10
True
>>> num = 11
>>> num >= 1 and num<=10
False
>>>
```



Una variables es un identificador que referencia a un valor. No hay que declarar la variable antes de usarla, el tipo de la variable será el mismo que el del valor al que hace referencia. Por lo tanto su tipo puede cambiar en cualquier momento:

>>> var = 5 >>> type(var) <class 'int'> >>> var = "hola" >>> type(var) <class 'str'>

Creación, borrado y ámbito de variables

Como hemos comentado anteriormente para crear una variable simplemente tenemos que utilizar un operador de asignación, el más utilizado = para que referencia un valor. Si queremos borrar la variable utilizamos la instrucción del. Por ejemplo:

>>> a = 5 >>> a 5 >>> del a >>> a Traceback (most recent call last): File "<stdin>", line 1, in <module> NameError: name 'a' is not defined
El ámbito de una variable se refiere a la zona del programa donde se ha definido y existe esa variable. Como primera aproximación las variables creadas dentro de funciones o clases tienen un ámbito local, es decir no existen fuera de la función o clase. Concretaremos cuando estudiamos estos aspectos más profundamente.

Modificación del valor de una variable

En cualquier momento podemos cambiar el valor de una variable, asignándole un nuevo valor:

>>> a = 5 >>> a 5 >>> a = 8 >>> a 8
También podemos modificar el valor de una variable, por ejemplo si queremos incrementarla en uno, podríamos usar:

>>> a = a + 1
Aunque también podemos utilizar otro operador de asignación:

>>> a+=1
Otros operadores de asignación: +=, -=, *=, /=, %=, **=, //=.

Función input

No permite leer por teclado información. Devuelve una cadena de caracteres y puede tener como argumento una cadena que se muestra en pantalla.

Ejemplos

>>> nombre=input("Nombre:") Nombre:jose >>> nombre 'jose' >>> edad=int(input("Edad:")) Edad:23 >>> edad 23
Función print

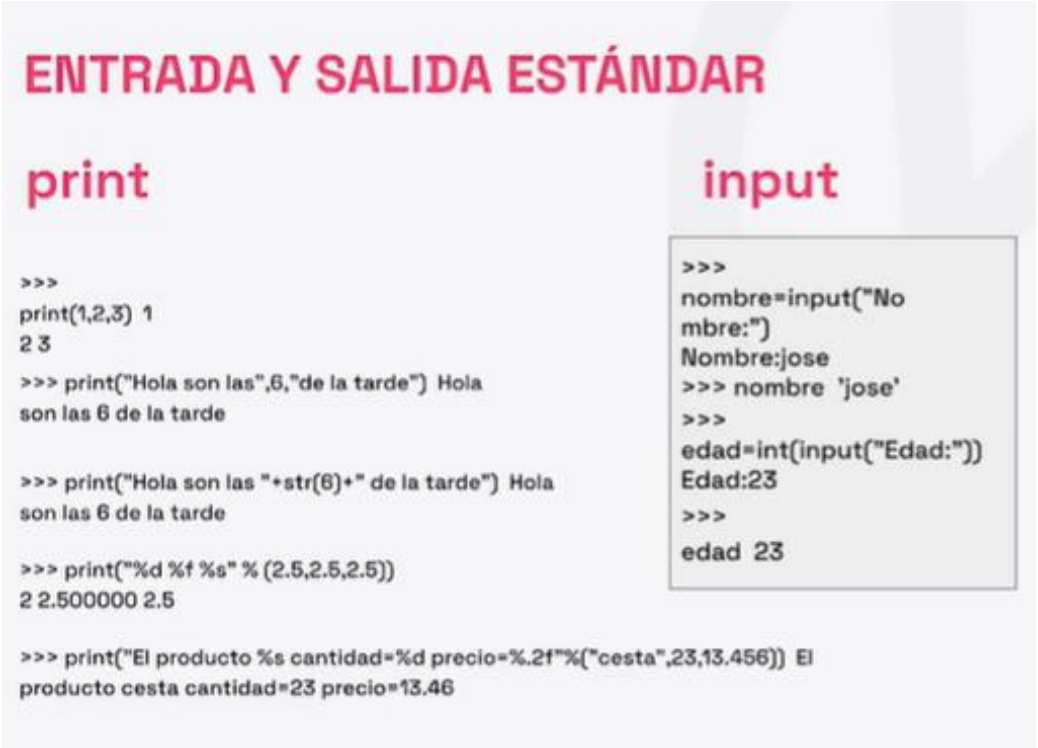
No permite escribir en la salida estándar. Podemos indicar varios datos a imprimir, que por defecto serán separado por un espacio. Podemos también imprimir varias cadenas de texto utilizando la concatenación.

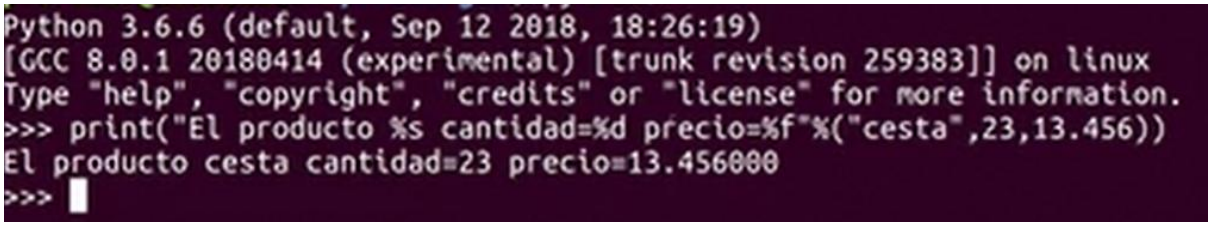
Ejemplos

>>> print(1,2,3) 1 2 3 >>> print("Hola son las",6,"de la tarde") Hola son las 6 de la tarde >>> print("Hola son las "+str(6)+" de la tarde") Hola son las 6 de la tarde
Formateando cadenas de caracteres

Con la función print Podemos indicar el formato con el que se va a mostrar los datos, por ejemplo:

>>> print("%d %f %s" % (2.5,2.5,2.5)) 2 2.500000 2.5 >>> print("El producto %s cantidad=%d precio=%.2f"%("cesta",23,13.456)) El producto cesta cantidad=23 precio=13.46





Las cadenas de caracteres (str): Me permiten guardar secuencias de caracteres.

Definición de cadenas

Podemos definir una cadena de caracteres de distintas formas:

>>> cad1 = "Hola" >>> cad2 = '¿Qué tal?' >>> cad3 = "'Hola, que tal?"'
Operaciones básicas con cadenas de caracteres

Algunas de las operaciones que puedo realizar con las cadenas de caracteres son:

- Concatenación: +: El operador + me permite unir datos de tipos secuenciales, en este caso dos cadenas de caracteres.

>>> "hola " + "que tal" 'hola que tal'

- Repetición: *: El operador * me permite repetir un dato de un tipo secuencial, en este caso de cadenas de caracteres.

>>> "abc" * 3 'abcabcabc'

- Indexación: Puedo obtener el dato de una secuencia indicando la posición en la secuencia. En este caso puedo obtener el carácter de la cadena indicando la posición (**empezando por la posición 0**).

>>> cadena = "josé" >>> cadena[0] 'j' >>> cadena[3] 'é'

- Para obtener la longitud de una cadena (número de caracteres que tiene), utilizamos la función len:

>>> cadena = "josé" >>> len(cadena) 4

Comparación de cadenas

Las cadenas se comparan carácter a carácter, en el momento en que dos caracteres no son iguales se compara alfabéticamente (es decir, se convierte a código unicode y se comparan).

Ejemplos

>>> "a">"A" True >>> "informatica">"informacion" True >>> "abcde">"abcdef" False

```
... como estás'''
>>> cad1
'hola'
>>> cad2
'hola 2'
>>> cad3
'hola\nque tal\ncomo estás'
>>> print(cad3)
hola
que tal
como estás
>>> cad1 + cad2
'holahola 2'
>>> cad4 = cad1+cad2
>>> cad4
'holahola 2'
>>> cad1 * 10
'holaholaholaholaholaholaholaholaholahola'
>>> cad5 = cad1 * 10
>>> cad5
'holaholaholaholaholaholaholaholaholahola'
```

Introducción a las cadenas de caracteres

```
>>> cad1 = "Hola"
>>> cad2 = '¿Qué tal?'
>>> cad3 = '''Hola,
que tal?'''
```

Comparación

```
"a">"A"
True

"informatica">"informacion
"
True

"abcde">"abcdef"
False
```

Operadores

+: Concatenación

```
>>> "hola " + "que tal"
'hola que tal'
```

***: Repetición**

```
>>> "abc" * 3
'abcabcabc'
```

Indexación

```
>>> cadena = "josé"
>>> cadena[0]
'j'
>>> cadena[3]
'é'
```

Longitud

```
>>> cadena = "josé"
>>> len(cadena)
4
```

Pasando de pseudocódigo a python3

Pseudocódigo

```
Definir nombre Como Cadena;
Escribir "Dime tu nombre:";
Leer nombre;
```

Python3

```
# nombre = input("Dime tu nombre:")
```

Pseudocódigo

```
Definir numero Como Entero;
Escribir "Dime un número entero:";
Leer numero;
```

Python3

```
numero = int(input("Dime un número entero:"))
```

Pseudocódigo

```
Definir numero Como Real;
Escribir "Dime un número real:";
Leer numero;
```

Python3

```
numero = float(input("Dime un número real:"))
```

Pasando de pseudocódigo a python3

Pseudocódigo
```
Escribir "Hola ",nombre;
```
Python3
```
print("Hola",nombre)
```

Pseudocódigo
```
Escribir Sin Saltar var," ";
```
Python3
```
print(var," ",end="")
```

Pseudocódigo
```
numero <- 7;
```
Python3
```
numero = 7
```

Pseudocódigo
```
trunc(7/2)
```
Python3
```
7 // 2
```

Pasando de pseudocódigo a python3

Pseudocódigo
```
raiz(9)
```
Python3
```
import math
math.sqrt(9)
```

Pseudocódigo
```
subcadena(cadena,0,0)
```
Python3
```
cadena[0]
```

Pseudocódigo
```
cadena3 <- concatenar(cadena1,cadena2)
```
Python3
```
cadena3 = cadena1 + cadena2
```

Pseudocódigo
```
cadena <- Mayusculas(cadena)
```
Python3
```
cadena = cadena.upper()
```

Antes de empezar hacer ejercicios con la estructura secuencial, vamos a ver como convertimos algunas instrucciones en pseudocódigo a python3:

Leer variables por teclado

Leer cadenas de caracteres

En pseudocódigo:

Definir nombre Como Cadena; Escribir "Dime tu nombre:"; Leer nombre;
En Python3:

nombre = input("Dime tu nombre:")

Leer números enteros

En pseudocódigo:

Definir numero Como Entero; Escribir "Dime un número entero:"; Leer numero;
En Python3:

numero = int(input("Dime un número entero:"))

Leer números reales

En pseudocódigo:

Definir numero Como Real; Escribir "Dime un número real:"; Leer numero;
En Python3:

numero = float(input("Dime un número real:"))
Escribir una variable en pantalla

En pseudocódigo:

Escribir "Hola ",nombre;
En Python3:

print("Hola",nombre)
Escribir sin saltar a otra línea

En pseudocódigo:

Escribir Sin Saltar var," ";
En Python3:

print(var," ",end="")
Asignar valor a una variable

En pseudocódigo:

numero <- 7;
En Python3:

numero = 7
Por ejemplo para incrementar el valor de una variable, en pseudocódigo:

num <- num + 1
En python3 lo podemos hacer de dos maneras:

num = num +1
O de esta forma:

num += 1
Calcular la parte entera de una división

En pseudocódigo:

trunc(7/2)
En Python3:

7 // 2
Calcular la raíz cuadrada

En pseudocódigo:

raiz(9)
En Python3:

import math math.sqrt(9)
Obtener el carácter de una cadena

En pseudocódigo:

subcadena(cadena,0,0)
En Python3:

cadena[0]
Unir dos cadenas de caracteres

En pseudocódigo:

cadena3 <- concatenar(cadena1,cadena2)
En Python3:

cadena3 = cadena1 + cadena2
Convertir una cadena a Mayúsculas

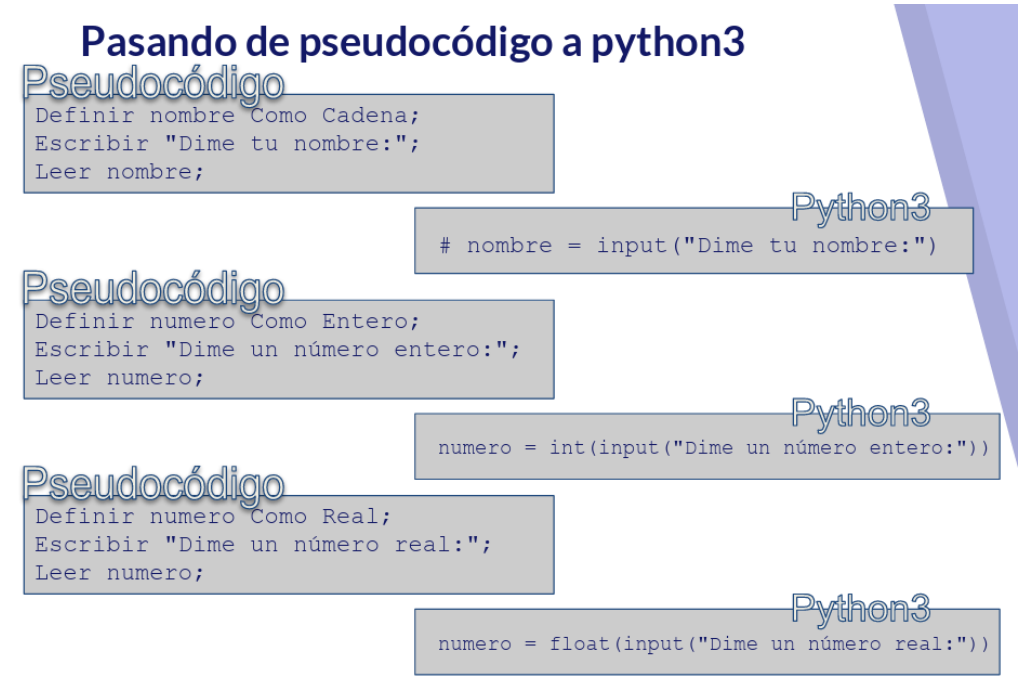
Lo veremos con detenimiento en las próximas unidades, pero vamos a usar el método de cadena upper:

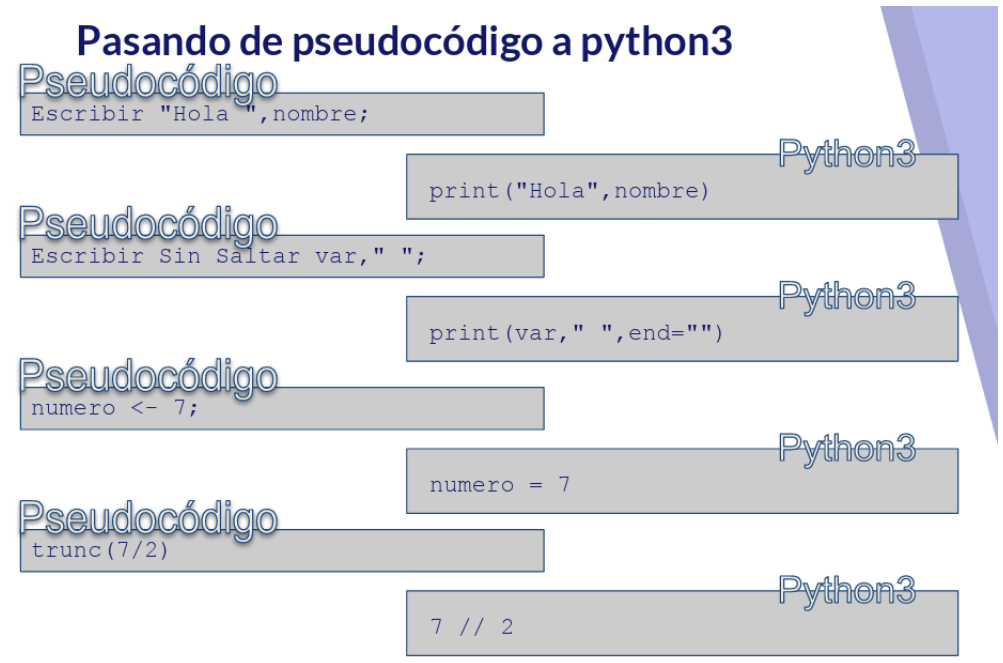
En pseudocódigo:

cadena <- Mayusculas(cadena)

En Python3:

cadena = cadena.upper()





Pasando de pseudocódigo a python3

Pseudocódigo

```
raiz(9)
```

Python3

```
import math
math.sqrt(9)
```

Pseudocódigo

```
subcadena(cadena,0,0)
```

Python3

```
cadena[0]
```

Pseudocódigo

```
cadena3 <- concatenar(cadena1,cadena2)
```

Python3

```
cadena3 = cadena1 + cadena2
```

Pseudocódigo

```
cadena <- Mayusculas(cadena)
```

Python3

```
cadena = cadena.upper()
```

Ejercicios estructura secuencial

Ejercicio 1

Escribir un programa que pregunte al usuario su nombre, y luego lo salude.

Ejercicio 2

Calcular el perímetro y área de un rectángulo dada su base y su altura.

Ejercicio 3

Dados los catetos de un triángulo rectángulo, calcular su hipotenusa.

Ejercicio 4

Dados dos números, mostrar la suma, resta, división y multiplicación de ambos.

Ejercicio 5

Escribir un programa que convierta un valor dado en grados Fahrenheit a grados Celsius. Recordar que la fórmula para la conversión es:

```
C = (F-32)*5/9
```

Ejercicio 6

Calcular la media de tres números pedidos por teclado.

Ejercicio 7

Realiza un programa que reciba una cantidad de minutos y muestre por pantalla a cuantas horas y minutos corresponde.
Por ejemplo: 1000 minutos son 16 horas y 40 minutos.

Ejercicio 8

Un vendedor recibe un sueldo base mas un 10% extra por comisión de sus ventas, el vendedor desea saber cuanto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.

Ejercicio 9

Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuanto deberá pagar finalmente por su compra.

Ejercicio 10

Un alumno desea saber cual será su calificación final en la materia de Algoritmos. Dicha calificación se compone de los siguientes porcentajes:

- 55% del promedio de sus tres calificaciones parciales.
- 30% de la calificación del examen final.
- 15% de la calificación de un trabajo final.

Ejercicio 11

Pide al usuario dos números y muestra la “distancia” entre ellos (el valor absoluto de su diferencia, de modo que el resultado sea siempre positivo).

Ejercicio 12

Pide al usuario dos pares de números x1,y2 y x2,y2, que representen dos puntos en el plano. Calcula y muestra la distancia entre ellos.

Ejercicio 13

Realizar un algoritmos que lea un número y que muestre su raíz cuadrada y su raíz cúbica. Python3 no tiene ninguna función predefinida que permita calcular la raíz cúbica, ¿Cómo se puede calcular?

Ejercicio 14

Dado un número de dos cifras, diseñe un algoritmo que permita obtener el número invertido. Ejemplo, si se introduce 23 que muestre 32.

Ejercicio 15

Dadas dos variables numéricas A y B, que el usuario debe teclear, se pide realizar un algoritmo que intercambie los valores de ambas variables y muestre cuanto valen al final las dos variables.

Ejercicio 16

Dos vehículos viajan a diferentes velocidades (v1 y v2) y están distanciados por una distancia d. El que está detrás viaja a una velocidad mayor. Se pide hacer un algoritmo para ingresar la distancia entre los dos vehículos (km) y sus respectivas velocidades (km/h) y con esto determinar y mostrar en que tiempo (minutos) alcanzará el vehículo más rápido al otro.

Ejercicio 17

Un ciclista parte de una ciudad A a las HH horas, MM minutos y SS segundos. El tiempo de viaje hasta llegar a otra ciudad B es de T segundos. Escribir un algoritmo que determine la hora de llegada a la ciudad B.

Ejercicio 18

Pedir el nombre y los dos apellidos de una persona y mostrar las iniciales.

Ejercicio 19

Escribir un algoritmo para calcular la nota final de un estudiante, considerando que: por cada respuesta correcta 5 puntos, por una incorrecta -1 y por respuestas en blanco 0. Imprime el resultado obtenido por el estudiante.

Ejercicio 20

Diseñar un algoritmo que nos diga el dinero que tenemos (en euros y céntimos) después de pedirnos cuantas monedas tenemos (de 2€, 1€, 50 céntimos, 20 céntimos o 10 céntimos).

Ejercicios resueltos

Ejercicios estructura secuencial

```
# Crea una aplicación que pida un número y calcule su factorial (El f
# un número es el producto de todos los enteros entre 1 y el propio n
# representa por el número seguido de un signo de exclamación.
# Por ejemplo 5! = 1x2x3x4x5=120)
```

Suma de todos los nº introducidos y la media de todos los números

```
# Algoritmo que pida números hasta que se introduzca un cero. Debe im
# y la media de todos los números introducidos.
```

Introducir un nº, como máximo 99 y mostrar la tabla de multiplicar de 1 a 10 de dicho número

```
# Realizar una algoritmo que muestre la tabla de multiplicar de un nú
# introducido por teclado.
```