

# TEMA 0.2: INTRO A PYTHON

Asignatura: Inteligencia Artificial

3º Curso Ingeniería Informática

Fecha de actualización: 21st August 2024

Prof. Dr. José Luis Salmerón



Escuela Politécnica Superior  
**CUNEF Universidad**



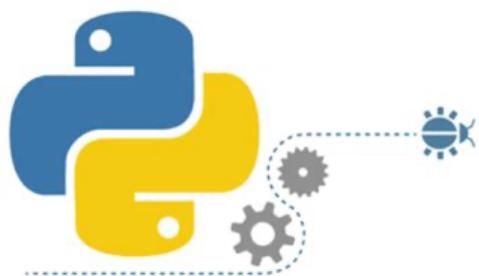
1. Entorno

2. Fundamentos

3. Colecciones

4. Control de flujo

5. Funciones



# ORIGEN

## Motivación y concepto

- Python es un lenguaje open source interpretado de scripts
- Lo desarrolló Guido Van Rossum en 1991
- Tomó su nombre por el grupo de cómicos ingleses Monty Python.
- Python es multiparadigma, soportando Orientación a Objetos, imperativa, funcional o procedural.



# ORIGEN

## Quien usa Python

- Google: Muchos componentes del buscador y el spider de Google están escritos en Python.
- NASA
- Yahoo
- Industrial Light & Magic
- Juegos como Battlefield 2, Civilization 4
- Walt Disney Feature Animation
- National Weather Service
- Red Hat
- Y muchos mas ...



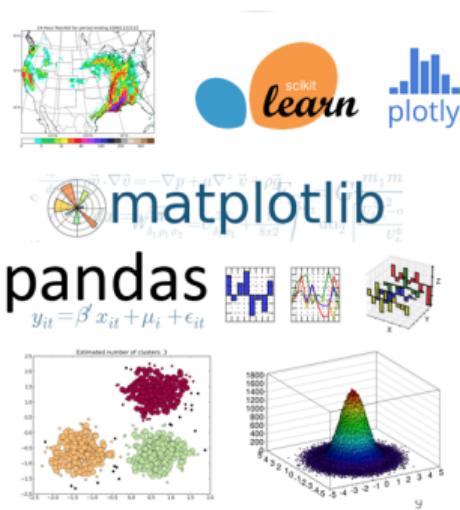
# DOCUMENTACIÓN

## Sites

- Completa:
  - ▶ Tutorial oficial: <http://docs.python.org/tutorial/>
  - ▶ Referencia del lenguaje: <http://docs.python.org/reference/>
- Novedades diarias y usuarios activos.
  - ▶ Pythonware Daily: <http://www.pythonware.com/daily/>
  - ▶ Planet Python: <http://planet.python.org/>
- ¡Respuestas a las dudas en segundos! ej.: [stackoverflow.com](http://stackoverflow.com)

## Librerías

- Scientific Computing: NumPy, SciPy, Pandas, IPython Notebook, ...
- Machine Learning: Scikit-learn, Shogun, PyLearn2, ...
- Plotting and Visualization: matplotlib, ggplot, plotly, ...



# VERSIONES

Python ha evolucionado significativamente desde su creación. Aquí se destacan algunas de las versiones más importantes:

- Python 2.x: Versión antigua, ampliamente utilizada en el pasado.
- Python 3.x: Versión actual con muchas mejoras y nuevas características.
- Python 3.8: Introducción de asignaciones de expresión (operador walrus :=).
- Python 3.9: Mejoras en las anotaciones de tipo y nuevos métodos en diccionarios.
- Python 3.10: Introducción de patrones de coincidencia estructurada.
- Python 3.11: Mejoras de rendimiento y características adicionales.
- Python 3.12: Versión actual. Mejoras en usabilidad.

# DISTRIBUCIONES

Las distribuciones de Python son versiones preempaquetadas de Python que incluyen el intérprete de Python, bibliotecas y herramientas. Facilitan la configuración rápida de un entorno de trabajo en Python.

- Distribución estándar de Python: es la versión oficial proporcionada por Python Software Foundation. Incluye el intérprete de Python y la biblioteca estándar.
- Anaconda: distribución popular para ciencia de datos y aprendizaje automático. Incluye una gran cantidad de bibliotecas científicas y herramientas como Jupyter Notebook y Conda.
- Miniconda: versión más ligera de Anaconda que solo incluye Conda, el administrador de paquetes y entornos. Permite a los usuarios instalar solo las bibliotecas que necesitan.
- ActivePython: distribución comercial de Python que incluye bibliotecas adicionales y soporte empresarial. Es proporcionada por ActiveState.
- WinPython: distribución de Python para Windows que es portátil y no requiere instalación. Incluye una serie de herramientas y bibliotecas útiles para el desarrollo en Python.
- Existen otras distribuciones especializadas de Python, como MicroPython para microcontroladores y PyPy, una implementación de Python con un rendimiento mejorado gracias a la compilación Just-In-Time (JIT).

## IDES

The screenshot shows the PyCharm interface. On the left is the project tree for 'django-tutorial-extended'. The code editor on the right displays the 'models.py' file:

```

import datetime
from django.db import models
from django.utils import timezone
from django.contrib import admin

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

    @admin.display(
        boolean=True,
        ordering="pub_date",
        description="Published recently?",
    )
    def was_published_recently(self):
        now = timezone.now()
        return now - datetime.timedelta(days=1) <= self.pub_date <= now

```

Figure: PyCharm

The screenshot shows the Visual Studio Code interface. The code editor on the left has 'example.py' open. The 'Variables' panel on the right shows the state of variables:

```

In [1]: v = (((df.A + df.B) * 1).cumprod()[-1])
Out[1]: 1
df: A
B

```

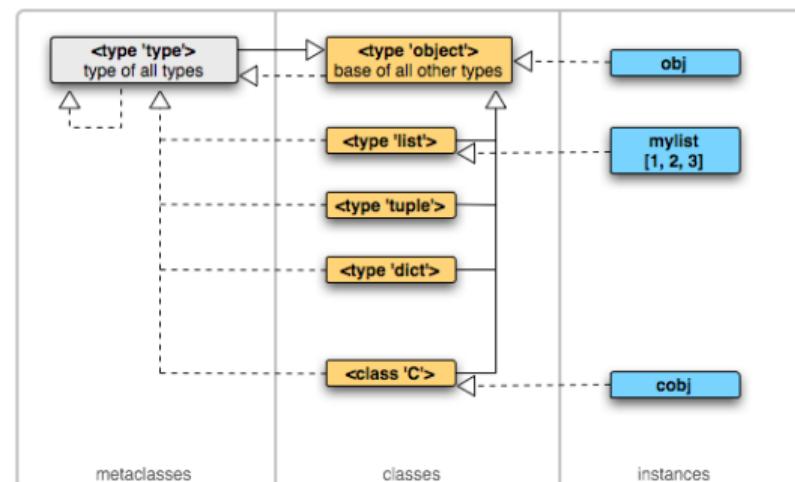
The code in 'example.py' includes imports for pandas, numpy, and matplotlib, and defines a DataFrame 'df' with some data.

Figure: Visual Studio Code

# ORIENTACIÓN A OBJETOS

## Todo en Python es un objeto

- Los objetos son entidades que tienen un determinado estado, comportamiento (método) e identidad.
- Los objetos tienen atributos (variables) y métodos (acciones/funciones).



# REPL

El REPL (Read-Eval-Print Loop) es un entorno interactivo que permite:

- Leer (Read) la entrada del usuario.
- Evaluar (Eval) la entrada como código Python.
- Imprimir (Print) el resultado de la evaluación.
- Repetir (Loop) el proceso.

Es una herramienta útil para experimentar y probar pequeños fragmentos de código de manera rápida.

## REPL mejorado

Jupyter Notebook o Jupyter Lab es un entorno computacional interactivo que combina la ejecución de código, texto enriquecido, plots etc.

Práctica:

1. \$ pip install jupyterlab
2. \$ jupyter lab

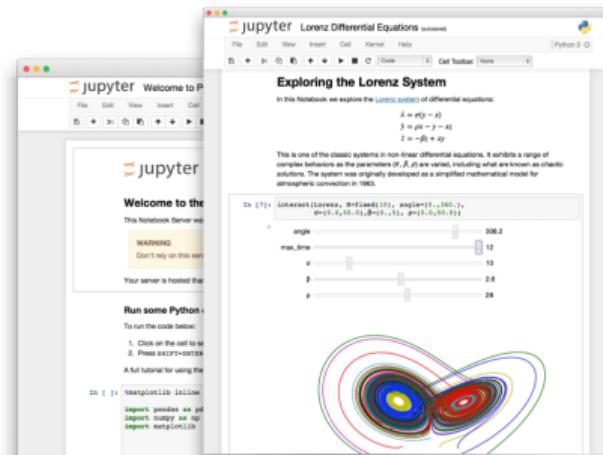
```

Python 3.6.3 (v3.6.3:2c5fed86e0, Oct  3 2017, 00:32:08)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: ls
20171107 Sesión 1/  20171121 Sesión 3/  ENV/
20171114 Sesión 2/  20171128 Sesión 4/  docs/

In [2]: print("Esto es iPython")
Esto es iPython

In [3]: 
```



# VERBOSIDAD

## Frente a otros lenguajes

- La sintaxis es user-friendly, consistente y elegante.
- Generalmente, el código Python es un 70% más corto que el correspondiente en Java.
- Python tiene una sintaxis más simple que otros lenguajes Orientados a Objetos.

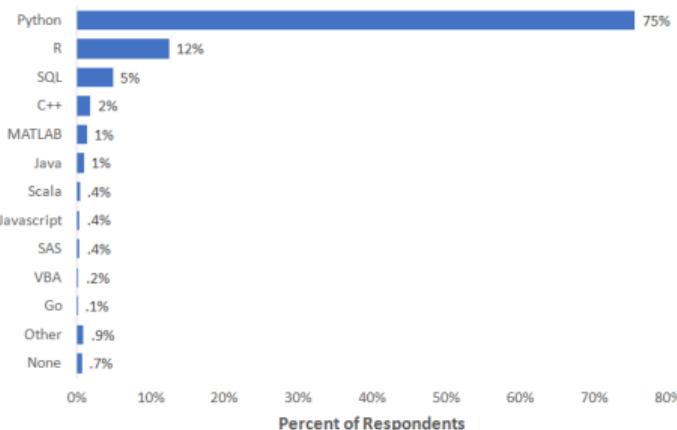
```
#include <stdio.h>
int main(void)
{
    printf("Hello, world!");
}
```

```
#include <iostream.h>
int main()
{
    std::cout << "Hello, world! ";
    return 0;
}
```

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello,
World!");
    }
}
```

C    C++    Java    Python

What programming language would you recommend an aspiring data scientist to learn first?



Note: Data are from the 2018 Kaggle ML and Data Science Survey. You can learn more about the study here: <http://www.kaggle.com/kaggle/kaggle-survey-2018>. A total of 23859 respondents completed the survey; the percentages in the graph are based on a total of 18788 respondents who provided an answer to this question.

## 1. Entorno

## 2. Fundamentos

## 2.1 Números y expresiones

## 2.2 Módulos

## 2.3 Tipos de datos

### 3. Colecciones

## 4. Control de flujo

## 5. Funciones



# SINTAXIS

## Constantes

- Las constantes numéricas son números. Los strings usan comillas simples ('') o dobles ("")
- La apariencia exacta del intérprete y sus mensajes de error dependerá de la versión que esté utilizando. Para ver si funciona, intente lo siguiente: `print("Hola mundo!")`
- En Python no es necesario finalizar cada línea con ningún símbolo como ; o cualquier otro. No obstante, si la añade al final no tendrá ningún efecto.
- En cambio si intenta ejecutar strings sin las comillas si recibirá un mensaje de error. El interprete indica donde está el error y de qué se trata.  
Es importante leer los mensajes de error.



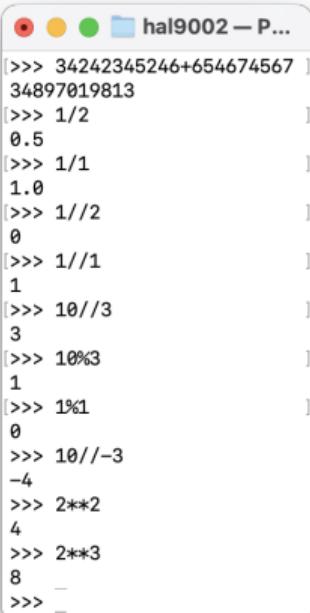
The screenshot shows a terminal window titled "hal9002 — Pyt...". It contains the following Python code and its execution:

```
>>> print("Hola mundo!")
Hola mundo!
>>> print("Hola mundo!");
[Hola mundo!
>>> Hola mundo
File "<stdin>", line 1
    Hola mundo
    ^^^^^^
SyntaxError: invalid syntax
>>> "Hola mundo"
'Hola mundo'
>>> _
```

# CONSTANTES

## Constantes

- El intérprete interactivo de Python se puede usar como una potente calculadora científica
- La división produce números decimales, llamados floats (coma flotante). La división trunca los enteros. La división de reales genera cocientes reales. En las operaciones con enteros y reales, el resultado es un número real. El entero se convierte en real antes de la operación. Para descartar la parte fraccionaria y hacer una división entera, puede usar una barra doble `<int>//<int>`
- El operador módulo `%` da el resto de `x` dividido por `y`. Es la parte que queda cuando usa la división de enteros. Es decir, `x % y` es igual que `x - ((x // y) * y)`. Redondea hacia abajo, por eso con valores negativos se aleja de cero.
- La potencia se calcula `x**y` o con la función `pow(x,y)`.



```
hal9002 — P...
>>> 34242345246+654674567
34897019813
>>> 1/2
0.5
>>> 1/1
1.0
>>> 1//2
0
>>> 1//1
1
>>> 10//3
3
>>> 10%3
1
>>> 1%1
0
>>> 10//−3
−4
>>> 2**2
4
>>> 2**3
8
>>> −
```

# VARIABLES

## Detalles sobre variables

- Una variable es un lugar etiquetado en la memoria, para salvar, acceder y actualizar los datos usando la etiqueta (o nombre) de la variable. El operador de asignación es = y la sintaxis <variable> = <expression>.
- No se puede usar antes de asignarle un valor, ya que no existe un valor por defecto. Los identificadores deben empezar con un carácter o un guión bajo, y admiten caracteres, números y guiones bajos.
- Por convención: camel case y guiones bajos n\_var. Case-sensitive
  - ▶ Correcto: spam eggs spam23 \_speed
  - ▶ Incorrecto: 23spam #sign var.12
  - ▶ Distintos: spam Spam SPAM

## Algunas palabras reservadas

- and continue except is print yield as assert def del exec if lambda not raise return True False None break class elif else import in or pass try while

A screenshot of a Mac OS X terminal window titled "HAL9000 — python — 39x29". The terminal displays several assignment statements:

```
>>> x
200
>>> spam = 'ads'
>>> __spam = 'emails'
>>> spam
'ads'
>>> __spam
'emails'
>>> 43spam = 234
File "<stdin>", line 1
    43spam = 234
                           ^
SyntaxError: invalid syntax
>>> #43spam = 234
...
>>> #43spam
...
>>> var.43 = 234
File "<stdin>", line 1
    var.43 = 234
                           ^
SyntaxError: invalid syntax
>>> x = x +2
...
>>> x
200
>>> x=x+2
>>> x
202
>>>
```

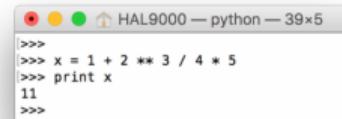
Two yellow callout boxes with black arrows point to specific parts of the code:

- A box labeled "Sentencia de asignación" points to the first assignment statement: `>>> spam = 'ads'`.
- A box labeled "Sentencia de asignación con una expresión" points to the third assignment statement: `>>> 43spam = 234`.

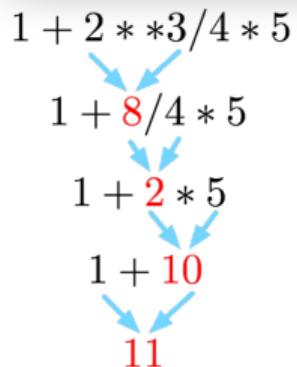
# PRECEDENCIA DE OPERADORES

## Constantes

- Cuando se ejecuta una expresión en Python con varios operadores, se debe saber el orden de ejecución de éstos
- Las reglas de precedencia de la más alta a la mas baja son:
  1. Paréntesis se respetan siempre
  2. Exponenciación
  3. Multiplicación, División y resto
  4. Suma y resta
  5. De izquierda a derecha
- Cuando se escriba código se recomienda el uso de paréntesis por legibilidad y evitar errores



```
>>>
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print x
11
>>>
```



# TIPOS

## Enteros y reales

- Al mezclar enteros y reales, el entero se convierte en real.
- Se puede controlar el tipo con funciones built-in **int()** y **float()**. Para conocer el tipo se usa **type()**. Se usan para convertir entre strings y números. Se obtiene un error si no tiene caracteres numéricos
- max()** valor máximo, **min()** mínimo y **abs()** valor absoluto.

```

>>> sval='123'
>>> type(sval)
<type 'str'>
>>> print sval + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> ival = int(sval)
>>> type(ival)
<type 'int'>
>>> print ival + 1
124
>>> nsv='hello bob'
>>> niv=int(nsv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hello bob'
>>> -

```

```

>>> xx =1
>>> type(xx)
<type 'int'>
>>> temp=9.4
>>> type(temp)
<type 'float'>
>>> type(1)
<type 'int'>
>>> type(1.0)
<type 'float'>
>>> print float(99)/100
0.99
>>> i=61
>>> type(i)
<type 'int'>
>>> f = float(i)
>>> print f
61.0
>>> type(f)
<type 'float'>
>>> print 1+2*float(3)/4-5
-2.5
>>>

```

```

In [2]: abs(-4)
Out[2]: 4

In [3]: max(1,2,4)
Out[3]: 4

In [4]: min(1,2,4)
Out[4]: 1

In [5]: max("Funciones")
Out[5]: 'u'

In [6]: min("Funciones")
Out[6]: 'F'

```

# ENTRADAS

## Entrada del usuario

- Con la función `input()` en Python 3.x se pausa y lee los datos del usuario. Esta función devuelve un string. Si se quiere un entero hay que utilizar una función de conversión
- Algunos operadores se puede aplicar los strings
  - ▶ `+` implica "concatenation"
  - ▶ `*` implica "multiple concatenation"
- Python sabe cuando se encuentra con un string o con un número y actúa en consecuencia. Para cambiar el tipo a string se usa la función `str()`

```
In [24]: name = input("¿Quién eres? ")  
¿Quién eres? Wenceslao  
In [25]: print("Hola", name)  
Hola Wenceslao  
In [26]: print('abc', 123)  
abc 123  
In [27]: type(123)  
Out[27]: int  
In [28]: type(str(123))  
Out[28]: str  
In [29]: print('Hola '*5)  
Hola Hola Hola Hola Hola  
In [30]:
```

# ENTRADAS

## Ejercicio

- Pida el nombre por consola e imprima un saludo usando su nombre
- Imprima un string y un entero
- Averigüe el tipo de un entero con la función `type()`, cambie dicho entero a string y averigüe su tipo
- Concatene un saludo cinco veces

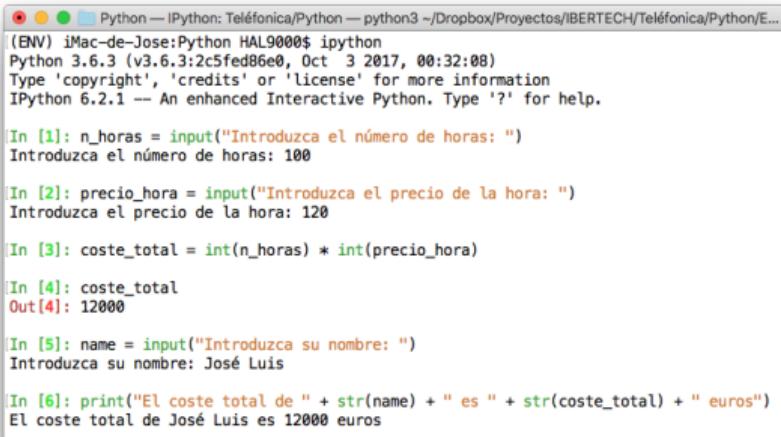
## Ejercicio

- Escriba un script en el terminal de Python que pida al usuario el número de horas, el precio de la hora y calcule el coste total
- Ahora incluye en el anterior que se le pida el nombre al usuario y de la siguiente salida  
`El coste total de <name> es <total> euros`

# ENTRADAS (SOLUCIÓN)

## Ejercicio

- Escriba un script en el terminal de Python que pida al usuario el número de horas, el precio de la hora y calcule el coste total
- Ahora incluye en el anterior que se le pida el nombre al usuario y de la siguiente salida  
**El coste total de <name> es <total> euros**



```
(ENV) iMac-de-Jose:Python HAL9000$ ipython
Python 3.6.3 (v3.6.3:2c5fed86e0, Oct  3 2017, 00:32:08)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: n_horas = input("Introduzca el número de horas: ")
Introduzca el número de horas: 100

In [2]: precio_hora = input("Introduzca el precio de la hora: ")
Introduzca el precio de la hora: 120

In [3]: coste_total = int(n_horas) * int(precio_hora)

In [4]: coste_total
Out[4]: 12000

In [5]: name = input("Introduzca su nombre: ")
Introduzca su nombre: José Luis

In [6]: print("El coste total de " + str(name) + " es " + str(coste_total) + " euros")
El coste total de José Luis es 12000 euros
```

# BUILT-IN

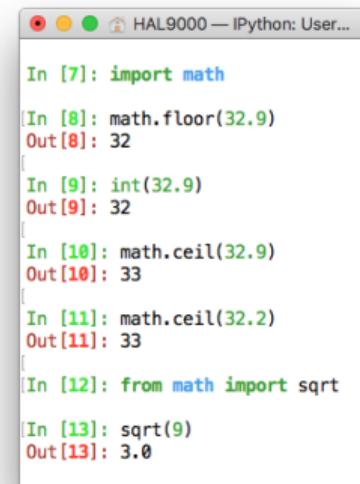
## Entrada del usuario

- Se importan módulos con el comando `import`.
- Si está seguro de que no importará más de una función con un nombre dado, es posible que no desee escribir el nombre del módulo cada vez que llame a la función. Puede usar:

```
from <module> import <function>
```

## Ejercicio

- Importe el módulo `math` y llame a la función `floor()` para redondear hacia abajo un número real (para esta operación se podría usar también `int()`).
- Llame a la función `math.ceil()` para redondear hacia arriba.
- Importe la función `sqrt()` del módulo `math` y calcule una raíz cuadrada.



The screenshot shows a Jupyter Notebook interface with the title "HAL9000 — IPython: User...". It displays the following code execution:

```
In [7]: import math
In [8]: math.floor(32.9)
Out[8]: 32
[...]
In [9]: int(32.9)
Out[9]: 32
[...]
In [10]: math.ceil(32.9)
Out[10]: 33
[...]
In [11]: math.ceil(32.2)
Out[11]: 33
[...]
In [12]: from math import sqrt
In [13]: sqrt(9)
Out[13]: 3.0
```

# CÓDIGO FUENTE

## Entrada del usuario

- Se importan módulos con el comando `import`.
- Si está seguro de que no importará más de una función con un nombre dado, es posible que no desee escribir el nombre del módulo cada vez que llame a la función. Puede usar: `from <module> import <function>`
- Los comentarios se inicián con `#` y comentan la línea entera

## Ejercicio

- Imprima un mensaje de saludo por consola.
- Capture una entrada por teclado del usuario donde se le solicite su nombre y edad.
- Imprima por pantalla un mensaje indicando la edad del usuario y su distancia a la media del rango de edad laboral.

## CÓDIGO FUENTE (SOLUCIÓN)

## Entrada del usuario

- Se importan módulos con el comando `import`.
  - Si no importa más de una función, es posible que no desee escribir el nombre del módulo cada vez que llame a la función. Puede usar: `from <module> import <function>`
  - Los comentarios se inician con `#` y comentan la línea a partir de donde se sitúan

# Ejercicio

Cree un fichero llamado **script1.py** que realice las siguientes acciones:

1. Imprima un mensaje de saludo por consola.
  2. Capture una entrada por teclado del usuario donde se le solicite su nombre y edad.
  3. Imprima por pantalla un mensaje indicando la edad del usuario y su distancia a la media del rango de edad laboral.

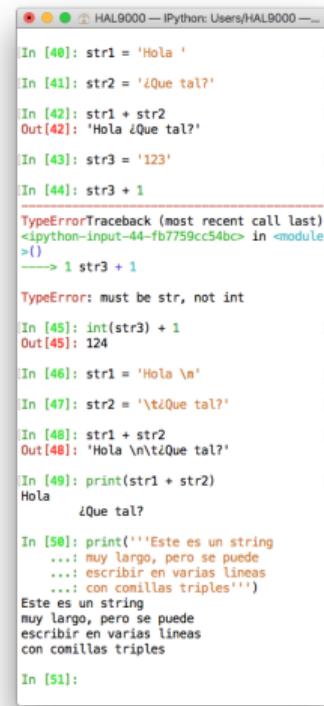
# CONCATENACIÓN Y TIPOS

## Concatenación y conversión de tipos

- Un string es una secuencia de caracteres y se usan comillas simples o dobles:  
'Hello' o "Hello"
- El operador + para strings implica concatenación
- Aunque un string contenga números sigue siendo un string. Es posible convertir números en un string en enteros usando int() o reales mediante float()
- \n genera un salto de línea y \t una tabulación. Para strings largos que abarcan varias líneas se pueden insertar con comillas triples.

## Ejemplo

1. Concatene dos strings
2. Concatene un string y un entero
3. Inserte un salto de línea y una tabulación
4. Inserte un string de varias líneas con retornos de carro con comillas triples.



```
In [40]: str1 = 'Hola '
In [41]: str2 = '\t¿Que tal?'
In [42]: str1 + str2
Out[42]: 'Hola \t¿Que tal?'
In [43]: str3 = '123'
In [44]: str3 + 1
TypeError: must be str, not int
In [45]: int(str3) + 1
Out[45]: 124
In [46]: str1 = 'Hola \n'
In [47]: str2 = '\t¿Que tal?'
In [48]: str1 + str2
Out[48]: 'Hola \n\t¿Que tal?'
In [49]: print(str1 + str2)
Hola
    ¿Que tal?
In [50]: print("""Este es un string
... muy largo, pero se puede
... escribir en varias lineas
... con comillas triples""")
Este es un string
muy largo, pero se puede
escribir en varias lineas
con comillas triples
In [51]:
```

# INDEXACIÓN

## Acceso y longitud

- Se puede extraer un carácter de un String usando un índice entre corchetes
- El índice debe ser un entero y empieza en cero
- El índice puede ser un valor calculado como resultado de una expresión
- Python devuelve un error si se intenta acceder a una posición más allá de la longitud del String
- La función built-in `len()` devuelve la longitud del String

## Ejemplo

1. Cree un string 'banana'
2. Realice pruebas de indexación
3. Extraiga la longitud



```
HAL9000 — IPython: Users/HAL9000...  
In [18]: fruit = 'banana'  
In [19]: fruit[1]  
Out[19]: 'a'  
  
In [20]: x = 3  
  
In [21]: fruit[x-1]  
Out[21]: 'n'  
  
In [22]: fruit[?]  
  
IndexErrorTraceback (most recent call last)  
-> <ipython-input-22-93efd9f4fb6> in <module>()  
----> 1 fruit[?]  
  
IndexError: string index out of range  
  
In [23]: len(fruit)  
Out[23]: 6  
  
In [24]: _
```

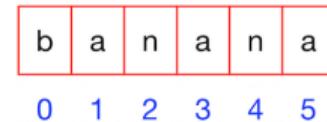
# INDEXACIÓN

## Acceso y longitud

- Hay una sintaxis personalizada para escribir cadenas que incluyen nuevas líneas (cadenas largas) o barras diagonales inversas (cadenas sin formato).
- Todas las cadenas en Python 3 son cadenas Unicode. Cada carácter Unicode se representa por su número en el estándar Unicode (<http://unicode-table.com>). Esto permite referirse a más de 120.000 caracteres en 129 sistemas de escritura. Existen mecanismos generales para especificar caracteres Unicode, ya sea por literales hexadecimales de 16 o 32 bits (prefijándolos con \u o \U, respectivamente) o por su Nombre Unicode (usando \N{nombre}).

## Ejemplo

1. Cree un string 'banana'
2. Realice pruebas de indexación
3. Extraiga la longitud



```

In [18]: fruit = 'banana'
In [19]: fruit[1]
Out[19]: 'a'
In [20]: x = 3
In [21]: fruit[x-1]
Out[21]: 'n'
In [22]: fruit[7]
IndexErrorTraceback (most recent call last)
<ipython-input-22-93efd9f4fbfb6> in <module>()
      1 fruit[7]
----> 1 fruit[7]

IndexError: string index out of range
In [23]: len(fruit)
Out[23]: 6
In [24]: -

```

A screenshot of an IPython terminal window. It shows the following session:

- In [18]: fruit = 'banana'
- In [19]: fruit[1]
- Out[19]: 'a'
- In [20]: x = 3
- In [21]: fruit[x-1]
- Out[21]: 'n'
- In [22]: fruit[7]
- IndexErrorTraceback (most recent call last)
<ipython-input-22-93efd9f4fbfb6> in <module>()
 1 fruit[7]
----> 1 fruit[7]
- IndexError: string index out of range
- In [23]: len(fruit)
- Out[23]: 6
- In [24]: -

# RESUMEN

## Funciones de utilidad

abs(number)	Devuelve el valor absoluto de un número
float(object)	Convierte una cadena o número a un número de coma flotante
help([object])	Ofrece ayuda interactiva
input(prompt)	Obtiene la entrada del usuario como una cadena
int(object)	Convierte una cadena o un número a un número entero
math.ceil(number)	Devuelve el techo de un número como real
math.floor(number)	Devuelve el techo de un número como real
math.sqrt(number)	Devuelve la raíz cuadrada; no funciona con números negativos
cmath.sqrt(number)	Devuelve la raíz cuadrada; funciona con números negativos
pow(x, y[, z])	Devuelve x a la potencia de y (modulo z)
print(object, ...)	Imprime los argumentos
round(number[, ndigits])	Redondea un número a una precisión dada, con enlaces redondeados al número par
str(object)	Convierte un valor en una cadena. Si convierte desde bytes, puede especificar el comportamiento de codificación y error

## 1. Entorno

## 2. Fundamentos

## 3. Colecciones

### 3.1 Listas

### 3.2 Tuplas

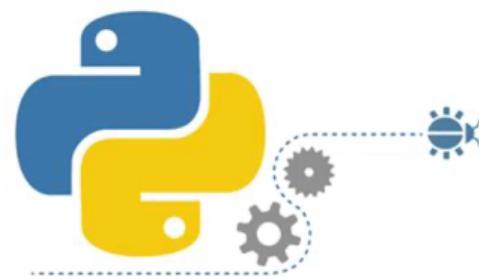
### 3.3 Strings

### 3.4 Diccionarios

### 3.5 Asignación

## 4. Control de flujo

## 5. Funciones



# CREACIÓN

## Creación de listas

- La estructura de datos más básica es la secuencia que se emplea con una colección de valores. A cada elemento de una secuencia se le asigna su posición o índice inicializado a cero.
- Python tiene varios tipos de secuencias integradas: listas, tuplas y strings.
- Una lista puede ser útil si necesita agregar elementos a medida que avanza. Los elementos de una lista están separados por comas y entre corchetes.
- Un método es una función que vinculada a algún objeto `object.method(arguments)`. Se puede crear una lista de un string con la función `list()`. Los string incluyen el método `<separador>.join(<string secuence>)`

## Ejemplo

1. Cree una tupla con los siguientes elementos `Mick Jagger, 79` y llámela `mick`
2. Cree una tupla con los siguientes elementos `Jaime García, 25` y llámela `jaime`
3. Cree una tupla que integre las dos tuplas anteriores
4. Cree una lista de strings
5. Una todos los elementos de la lista anterior en un string

```
HAL9000 — IPython: Users/HAL9000 — ipython — 53...
In [46]: mick = ['Mick Jagger', 79]
In [47]: jaime = ['Jaime García', 25]
In [48]: database = [mick, jaime]
In [49]: database
Out[49]: [['Mick Jagger', 79], ['Jaime García', 25]]
In [50]: list('String')
Out[50]: ['S', 't', 'r', 'i', 'n', 'g']
In [51]: a_list = list('String')
In [52]: ''.join(a_list)
Out[52]: 'String'
```

# INDEXACIÓN

## Acceso

- Todos los elementos en una secuencia están numerados, desde cero en adelante. Se accede a ellos individualmente mediante el índice entre corchetes `<lista>[<indice>]`. Recuerde que un string no es más que una lista de caracteres.
- Los índices negativos inicia el recorrido en sentido inverso.
- Las listas (y secuencias en general) se pueden indexar directamente, sin usar una variable para referirse a ellos.
- Si una llamada a función devuelve una secuencia, puede indexarla directamente.

```
HAL9000 — IPython:...  
In [65]: str = 'Carlos Puig'  
In [66]: str[0]  
Out[66]: 'C'  
  
In [67]: str[-1]  
Out[67]: 'g'  
  
In [68]: 'José Luis'[1]  
Out[68]: 'o'  
  
In [69]: input('Año: ')[3]  
Año: 2017  
Out[69]: '7'  
  
In [70]:
```

## Ejemplo

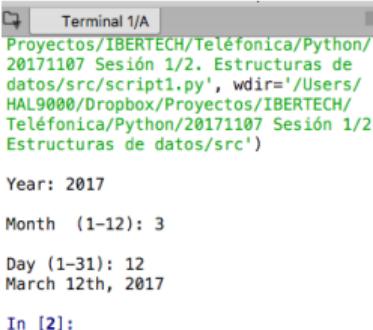
1. Cree un string '`Carlos Puig`'
2. Acceda al primer carácter y al último
3. Capture por teclado un año y extraiga únicamente el último dígito

# EJERCICIO 1 LISTAS

## Ejemplo

1. Cree un script que pida las fechas en números como se ve en la imagen a la derecha y devuelva la fecha en formato textual.
2. Tip: lista con los meses en inglés

```
'January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October',  
'November', 'December'
```



```
Terminal 1/A  
Proyectos/IBERTECH/Teléfonica/Python/  
20171107 Sesión 1/2. Estructuras de  
datos/src/script1.py', wdir='/Users/  
HAL9000/Dropbox/Proyectos/IBERTECH/  
Teléfonica/Python/20171107 Sesión 1/2.  
Estructuras de datos/src')  
  
Year: 2017  
Month (1-12): 3  
Day (1-31): 12  
March 12th, 2017  
  
In [2]:
```

## EJERCICIO 1 LISTAS (SOLUCIÓN)

# SLICING

## Acceso múltiple

- Se puede acceder a rangos. Usa dos índices, separados por dos puntos [`<start>:<end>`]. Si uno de los dos índices no se incluye se extiende hasta el extremo.
- Tened en cuenta que el primer índice se incluye y el segundo se excluye de la selección. Si los índices están fuera de la lista no se incluyen pero no devuelve error.

## Ejemplo: String con una etiqueta HTML

1. Extraiga el URL con un rango de índices y el texto con un rango de índices negativos
2. Cree una lista con números del 1 al 10
3. Extraiga los números 4 al 6
4. Extraiga con un rango de índices el primer elemento
5. Extraiga los elementos que ocupan los índices 7 a 10 y de 7 a 11.
6. Extraiga los números 8 y 9 con índices negativos
7. Los tres últimos números, tenga la longitud que tenga la lista
8. Toda la lista salvo los tres primeros
9. Toda la lista con un rango de índices

# SLICING (SOLUCIÓN 1)

## Acceso múltiple

- Se puede acceder a rangos. Usa dos índices, separados por dos puntos [`<start>:<end>`]. Si uno de los dos índices no se incluye se extiende hasta el extremo.
- Tened en cuenta que el primer índice se incluye y el segundo se excluye de la selección. Si los índices están fuera de la lista no se incluyen pero no devuelve error.

## Ejemplo: String con una etiqueta HTML

1. Extraiga el URL con un rango de índices y el texto con un rango de índices negativos
2. Cree una lista con números del 1 al 10
3. Extraiga los números 4 al 6
4. Extraiga con un rango de índices el primer elemento
5. Extraiga los elementos que ocupan los índices 7 a 10 y de 7 a 11.
6. Extraiga los números 8 y 9 con índices negativos
7. Los tres últimos números, tenga la longitud que tenga la lista
8. Toda la lista salvo los tres primeros
9. Toda la lista con un rango de índices

```
HAL9000 — iPython: Users/HAL9000 — ipython3 — 54...
In [71]: tag = '<a href="http://www.python.org">Python
...: web site</a>'

In [72]: tag[9:38]
Out[72]: 'http://www.python.org'

In [73]: tag[32:-4]
Out[73]: 'Python web site'

In [74]: numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [75]: numbers[3:6]
Out[75]: [4, 5, 6]

In [76]: numbers[0:1]
Out[76]: [1]

In [77]: numbers[7:10]
Out[77]: [8, 9, 10]

In [78]: numbers[7:11]
Out[78]: [8, 9, 10]

In [79]: numbers[-3:-1]
Out[79]: [8, 9]

In [80]: numbers[-3:0]
Out[80]: []

In [81]: numbers[-3:]
Out[81]: [8, 9, 10]

In [82]: numbers[3:]
Out[82]: [4, 5, 6, 7, 8, 9, 10]

In [83]: numbers[::]
Out[83]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [84]: _
```

# SLICING

## Acceso múltiple

- Del mismo modo que usa la indexación para acceder a elementos individuales, se puede acceder a rangos de elementos.
- Para ello, use dos índices, separados por dos puntos [`<start>:<end>`]. Si uno de los dos índices no se incluye se extiende hasta el extremo.
- Hay que tener en cuenta que el primer índice se incluye y el segundo se excluye de la selección.
- Si los índices están fuera de la lista no se incluyen pero no devuelve error.

## Ejemplo 2

1. Escriba un script que le pide una URL y (suponiendo que sea de la forma limitada `http://www.somedomainname.com`) extraiga el nombre de dominio.

# SLICING (SOLUCIÓN 2)

## Acceso múltiple

- Del mismo modo que usa la indexación para acceder a elementos individuales, se puede acceder a rangos de elementos.
- Para ello, use dos índices, separados por dos puntos [`<start>:<end>`]. Si uno de los dos índices no se incluye se extiende hasta el extremo.
- Hay que tener en cuenta que el primer índice se incluye y el segundo se excluye de la selección.
- Si los índices están fuera de la lista no se incluyen pero no devuelve error.

```
9 # Pide una URL con formato
10 # http://www.somedomainname.com
11 # y extraiga el nombre de dominio.
12
13 # (1) Captura el URL
14 url = input('Please enter the URL:')
15
16 # (2) Extrae el dominio
17 domain = url[11:-4]
18
19 # (3) Muestra el dominio
20 print("Nombre de dominio: " + domain)
```

## Ejemplo 2

1. Escriba un script que le pide una URL y (suponiendo que sea de la forma limitada `http://www.somedomainname.com`) extraiga el nombre de dominio.

# SLICING

## Secuencias con diferentes steps

- Para incrementos entre elementos de una secuencia distintos de 1, se emplea la sintaxis [`<start>:<end>:<step>`]. Si el valor `<step>` es negativo los saltos serán hacia atrás
- La secuencias se pueden concatenar con el operador +

## Multiplicando secuencias

- Multiplicar una secuencia por un número `x` crea una nueva secuencia donde la secuencia original se repite `x` veces.

## Ejemplo 3

- Se muestran diferentes extracciones de la secuencia `numbers` con `steps`.
- Concatene un lista de enteros y dos strings.
- Multiplique una lista de enteros y un string por 2

```
● ○ □ HAL9000 — IPython: Users/HAL9000 ...
In [86]: numbers[0:10:1]
Out[86]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
In [87]: numbers[0:10:2]
Out[87]: [1, 3, 5, 7, 9]
In [88]: numbers[3:6:3]
Out[88]: [4]
In [89]: numbers[::-1]
Out[89]: [1, 5, 9]
In [90]: numbers[8:3:-1]
Out[90]: [9, 8, 7, 6, 5]
In [91]: numbers[10:0:-2]
Out[91]: [10, 8, 6, 4, 2]
In [92]: numbers[0:10:-2]
Out[92]: []
In [93]: numbers[::-2]
Out[93]: [10, 8, 6, 4, 2]
In [94]: numbers[5::-2]
Out[94]: [6, 4, 2]
In [95]: numbers[::-2]
Out[95]: [10, 8]
```

```
● ○ □ HAL9000 — IPython: Users/...
In [1]: [1, 2, 3] + [4, 5, 6]
Out[1]: [1, 2, 3, 4, 5, 6]
In [2]: 'Hola ' + 'mundo'
Out[2]: 'Hola mundo'
```

```
● ○ □ HAL9000 — IPython: Us...
In [13]: [1,3,5] * 2
Out[13]: [1, 3, 5, 1, 3, 5]
In [14]: [3] * 2
Out[14]: [3, 3]
In [15]: ['Hola'] * 2
Out[15]: ['Hola', 'Hola']
In [16]:
```

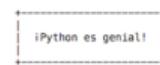
# NONE, LISTAS VACÍAS E INICIALIZACIÓN

## None

- Una lista vacía se escribe con dos corchetes [] .
- Para crear una lista con la habitación para diez elementos pero sin nada se emplearía **None**, que es un valor que significa: "no hay nada".
- **nan** si es tipo de dato que permite algunas operaciones.

```
In [1]: runfile('/Users/HAL9000/Dropbox/Proyectos/IBERTECH/Teléfonica/  
Python/20171107 Sesión 1/2. Estructuras de datos/src/script4.py', wdir='/  
Users/HAL9000/Dropbox/Proyectos/IBERTECH/Teléfonica/Python/20171107 Sesión  
1/2. Estructuras de datos/src')
```

Sentence: iPython es genial!



## Ejemplo

1. Capture una frase por consola
2. Imprímala por consola con un cuadro alrededor con un ancho ajustable tal y como se muestra en la imagen

# NONE, LISTAS VACÍAS E INICIALIZACIÓN (SOLUCIÓN)

## None

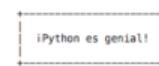
- Una lista vacía se escribe con dos corchetes [] .
- Para crear una lista con la habitación para diez elementos pero sin nada se emplearía `None`, que es un valor que significa: "no hay nada".
- `nan` si es tipo de dato que permite algunas operaciones.

## Ejemplo

1. Capture una frase por consola
2. Imprímala por consola con un cuadro alrededor con un ancho ajustable tal y como se muestra en la imagen

```
In [1]: runfile('/Users/HAL9000/Dropbox/Proyectos/IBERTECH/Teléfonica/Python/20171107 Sesión 1/2. Estructuras de datos/src/script4.py', wdir='/Users/HAL9000/Dropbox/Proyectos/IBERTECH/Teléfonica/Python/20171107 Sesión 1/2. Estructuras de datos/src')
```

Sentence: iPython es genial!



```
9 # Prints a sentence in a centered "box" of correct width
10
11 sentence = input("Sentence: ")
12
13 screen_width = 80
14 text_width = len(sentence)
15 box_width = text_width + 6
16 left_margin = (screen_width - box_width) // 2
17
18 print()
19 print(' ' * left_margin + '+' + '-' * (box_width-2) + '+')
20 print(' ' * left_margin + '[' + ' ' * text_width + ']')
21 print(' ' * left_margin + '[' + sentence + ']')
22 print(' ' * left_margin + '[' + ' ' * text_width + ']')
23 print(' ' * left_margin + '[' + '-' * (box_width-2) + ']')
24 print()
```

# FUNCIONES BUILT-IN Y PERTENENCIA

## Funciones built-in

- La función `len` devuelve la cantidad de elementos que contiene una secuencia. `min` y `max` devuelven los elementos máximos y mínimos de la secuencia, admitiendo un número variable de argumentos.

```
In [32]: numbers = [100, 34, 678]
In [33]: len(numbers)
Out[33]: 3
In [34]: max(numbers)
Out[34]: 678
In [35]: min(numbers)
Out[35]: 34
In [36]: max(3,6,1,9)
Out[36]: 9
In [37]: min(3,6,1,9)
Out[37]: 1
```

## in

- Para verificar si se puede encontrar un valor en una secuencia, use el operador `in` (la no pertenencia se verifica con `not in`).
- Devuelve un valor booleano: `True` o `False`.

## Ejemplo

- Cree un lista cuyos elementos sean una lista que contenga un usuario y un código PIN
- Compruebe si un usuario y su PIN se encuentran en dicha lista

```
In [27]: database = [
...:     ['albert', '1234'],
...:     ['dilbert', '4242'],
...:     ['smith', '7524'],
...:     ['jones', '9843'],
...: ]
In [28]: ['smith', '7524'] in database
Out[28]: True
```

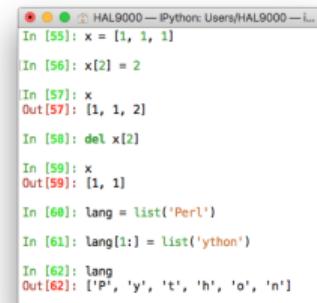
# OPERACIONES CON LISTAS

## Actualización

- Los elementos de las listas se pueden actualizar utilizando el operador de asignación = en la posición que se desea actualizar. Se pueden actualizar varios elementos con slices.
- Para eliminar un elemento se emplea el comando `del`

## Ejemplo

- Cree una lista con tres 1 y actualice el tercer elemento a 2
- Elimine el tercer elemento
- Cree una lista con el string 'Perl' y llámelo `lang`
- Actualice `lang` y cambie el contenido a 'Python'



```

HAL9000 — IPython: Users/HAL9000 — i...
In [55]: x = [1, 1, 1]
In [56]: x[2] = 2
In [57]: x
Out[57]: [1, 1, 2]
In [58]: del x[2]
In [59]: x
Out[59]: [1, 1]
In [60]: lang = list('Perl')
In [61]: lang[1:] = list('ython')
In [62]: lang
Out[62]: ['P', 'y', 't', 'h', 'o', 'n']

```

## Ejercicio

- Cree una lista llamada `numbers` que contenga los números 1 y 5
- Inserte mediante slices los números 2, 3 y 4
- Elimine mediante slices los números 2, 3 y 4

# OPERACIONES CON LISTAS (SOLUCIÓN)

## Actualización

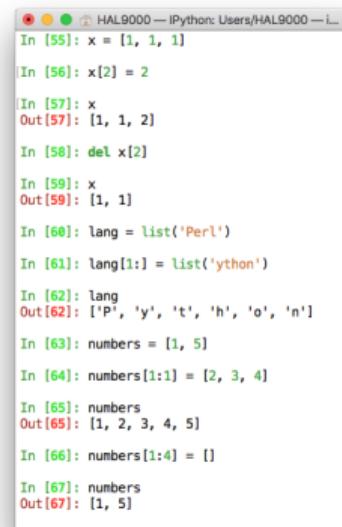
- Los elementos de las listas se pueden actualizar utilizando el operador de asignación = en la posición que se desea actualizar. Se pueden actualizar varios elementos con slices.
- Para eliminar un elemento se emplea el comando `del`

## Ejemplo

- Cree una lista con tres 1 y actualice el tercer elemento a 2
- Elimine el tercer elemento
- Cree una lista con el string 'Perl' y llámelo `lang`
- Actualice `lang` y cambie el contenido a 'Python'

## Ejercicio

- Cree una lista llamada `numbers` que contenga los números 1 y 5
- Inserte mediante slices los números 2, 3 y 4
- Elimine mediante slices los números 2, 3 y 4



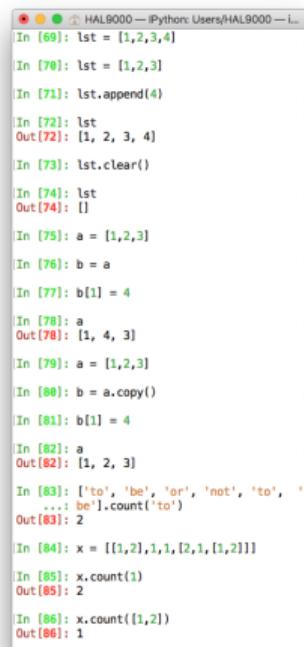
```
In [55]: x = [1, 1, 1]
In [56]: x[2] = 2
In [57]: x
Out[57]: [1, 1, 2]
In [58]: del x[2]
In [59]: x
Out[59]: [1, 1]
In [60]: lang = list('Perl')
In [61]: lang[1:] = list('ython')
In [62]: lang
Out[62]: ['P', 'y', 't', 'h', 'o', 'n']
In [63]: numbers = [1, 5]
In [64]: numbers[1:1] = [2, 3, 4]
In [65]: numbers
Out[65]: [1, 2, 3, 4, 5]
In [66]: numbers[1:4] = []
In [67]: numbers
Out[67]: [1, 5]
```

# MÉTODOS DE LISTAS

- <lista>.append(<elementos>) añade elementos al final
- <lista>.clear() elimina todos los elementos
- <lista>.copy() copia la lista. Recuerda: la asignación asigna otro nombre a la misma lista.
- <lista>.count(<elemento>) cuenta las ocurrencias de un elemento
- <lista>.extend(<lista>) extiende una lista con otra. Si se concatenan con + la lista original no se modifica
- <lista>.index(<elemento>) devuelve el índice de un elemento
- <lista>.insert(<índice>,<elemento>) devuelve el índice de un elemento
- <lista>.pop([<índice>]) elimina un elemento en una lista. Por defecto el último
- <lista>.remove(<elemento>) elimina la primera ocurrencia de un elemento
- <lista>.reverse() invierte el orden
- <lista>.sort() ordena una lista. No devuelve nada por lo que no se puede asignar. Si se desea una copia ordenada de una lista se puede usar la función sorted(<lista>)

## Ejemplo

1. Pruebe todas las funciones



```

In [69]: lst = [1,2,3,4]
In [70]: lst = [1,2,3]
In [71]: lst.append(4)
In [72]: lst
Out[72]: [1, 2, 3, 4]
In [73]: lst.clear()
In [74]: lst
Out[74]: []
In [75]: a = [1,2,3]
In [76]: b = a
In [77]: b[1] = 4
In [78]: a
Out[78]: [1, 4, 3]
In [79]: a = [1,2,3]
In [80]: b = a.copy()
In [81]: b[1] = 4
In [82]: a
Out[82]: [1, 2, 3]
In [83]: ['to', 'be', 'or', 'not', 'to', ...: be'].count('to')
Out[83]: 2
In [84]: x = [[1,2],1,1,[2,1,1,2,2]]
In [85]: x.count(1)
Out[85]: 2
In [86]: x.count([1,2])
Out[86]: 1

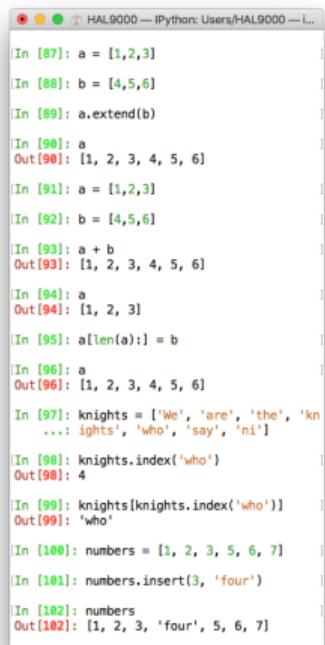
```

# MÉTODOS DE LISTAS

- <lista>.append(<elementos>) añade elementos al final
- <lista>.clear() elimina todos los elementos
- <lista>.copy() copia la lista. Recuerda: la asignación asigna otro nombre a la misma lista.
- <lista>.count(<elemento>) cuenta las ocurrencias de un elemento
- <lista>.extend(<lista>) extiende una lista con otra. Si se concatenan con + la lista original no se modifica
- <lista>.index(<elemento>) devuelve el índice de un elemento
- <lista>.insert(<índice>,<elemento>) devuelve el índice de un elemento
- <lista>.pop([<índice>]) elimina un elemento en una lista. Por defecto el último
- <lista>.remove(<elemento>) elimina la primera ocurrencia de un elemento
- <lista>.reverse() invierte el orden
- <lista>.sort() ordena una lista. No devuelve nada por lo que no se puede asignar. Si se desea una copia ordenada de una lista se puede usar la función sorted(<lista>)

## Ejemplo

1. Pruebe todas las funciones



```
In [87]: a = [1,2,3]
In [88]: b = [4,5,6]
In [89]: a.extend(b)
In [90]: a
Out[90]: [1, 2, 3, 4, 5, 6]
In [91]: a = [1,2,3]
In [92]: b = [4,5,6]
In [93]: a + b
Out[93]: [1, 2, 3, 4, 5, 6]
In [94]: a
Out[94]: [1, 2, 3]
In [95]: a[len(a):] = b
In [96]: a
Out[96]: [1, 2, 3, 4, 5, 6]
In [97]: knights = ['We', 'are', 'the', 'kn ... ights', 'who', 'say', 'ni']
In [98]: knights.index('who')
Out[98]: 4
In [99]: knights[knights.index('who')]
Out[99]: 'who'
In [100]: numbers = [1, 2, 3, 5, 6, 7]
In [101]: numbers.insert(3, 'four')
In [102]: numbers
Out[102]: [1, 2, 3, 'four', 5, 6, 7]
```

# MÉTODOS DE LISTAS

- <lista>.append(<elementos>) añade elementos al final
- <lista>.clear() elimina todos los elementos
- <lista>.copy() copia la lista. Recuerda: la asignación asigna otro nombre a la misma lista.
- <lista>.count(<elemento>) cuenta las ocurrencias de un elemento
- <lista>.extend(<lista>) extiende una lista con otra. Si se concatenan con + la lista original no se modifica
- <lista>.index(<elemento>) devuelve el índice de un elemento
- <lista>.insert(<índice>, <elemento>) devuelve el índice de un elemento
- <lista>.pop([<índice>]) elimina un elemento en una lista. Por defecto el último
- <lista>.remove(<elemento>) elimina la primera ocurrencia de un elemento
- <lista>.reverse() invierte el orden
- <lista>.sort() ordena una lista. No devuelve nada por lo que no se puede asignar. Si se desea una copia ordenada de una lista se puede usar la función sorted(<lista>)

## Ejemplo

1. Pruebe todas las funciones

```
In [100]: x = [1, 2, 3]
In [101]: x.pop()
Out[101]: 3
In [102]: x
Out[102]: [1, 2]
In [103]: x.pop(0)
Out[103]: 1
In [104]: x
Out[104]: [2]
In [105]: x = [1, 2, 3]
In [106]: x.append(x.pop())
In [107]: x
Out[107]: [1, 2, 3]
In [108]: x = [1, 2, 3, 4, 5, 6]
In [109]: x.remove('be')
In [110]: x
Out[110]: [1, 2, 3, 4, 5]
In [111]: x.reverse()
In [112]: x
Out[112]: [5, 4, 3, 2, 1]
In [113]: x = [4, 6, 2, 1, 7, 9]
In [114]: x.sort()
In [115]: x
Out[115]: [1, 2, 4, 6, 7, 9]
In [116]: x = [1, 2, 3, 4, 5, 6, 7, 8]
In [117]: x
Out[117]: [1, 2, 3, 4, 5, 6, 7, 8]
In [118]: x.reverse()
In [119]: x
Out[119]: [8, 7, 6, 5, 4, 3, 2, 1]
In [120]: y = x.sort()
In [121]: print(y)
None
In [122]: x
Out[122]: [1, 2, 3, 4, 5, 6, 7, 8]
In [123]: y = x.copy()
In [124]: y.sort()
In [125]: x
Out[125]: [1, 2, 3, 4, 5, 6, 7, 8]
In [126]: y
Out[126]: [1, 2, 4, 6, 7, 9]
In [127]: sorted(x)
Out[127]: [1, 2, 4, 6, 7, 9]
```

# SORTING AVANZADO

## .sort()

- El método `<lista>.sort(key=<function>, reverse=<boolean>)` toma dos argumentos opcionales: **key** y **reverse**. Si desea utilizarlos, normalmente se especifican por su nombre.
- Al argumento **key** se le proporciona una función y se usa en el proceso de clasificación.
- El argumento **reverse** es booleano e invierte la función si se le asigna **True**. Por defecto es **False**.

## Ejemplo

1. Cree la lista 'aardvark', 'abalone', 'acme', 'add', 'aerate'
2. Ordene la lista en función de su longitud
3. Cree la lista 4, 6, 2, 1, 7, 9
4. Ordénela en orden inverso

# SORTING AVANZADO (SOLUCIÓN)

## .sort()

- El método `<lista>.sort(key=<function>, reverse=<boolean>)` toma dos argumentos opcionales: **key** y **reverse**. Si desea utilizarlos, normalmente se especifican por su nombre.
- Al argumento **key** se le proporciona una función y se usa en el proceso de clasificación.
- El argumento **reverse** es booleano e invierte la función si se le asigna **True**. Por defecto es **False**.



```
HAL9000 — iPython: Users/HAL9000 — ipython — 68x14
In [138]: x = ['aardvark', 'abalone', 'acme', 'add', 'aerate']
In [139]: x.sort(key=len)
In [140]: x
Out[140]: ['add', 'acme', 'aerate', 'abalone', 'aardvark']
In [141]: x = [4, 6, 2, 1, 7, 9]
In [142]: x.sort(reverse=True)
In [143]: x
Out[143]: [9, 7, 6, 4, 2, 1]
```

## Ejemplo

- Cree la lista 'aardvark', 'abalone', 'acme', 'add', 'aerate'
- Ordene la lista en función de su longitud
- Cree la lista 4, 6, 2, 1, 7, 9
- Ordénela en orden inverso

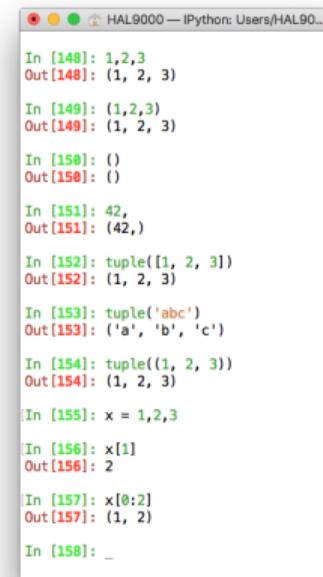
# SECUENCIAS INMUTABLES

## Creación y acceso a tuplas

- Las tuplas son secuencias, como las listas. La diferencia es que las tuplas no se pueden cambiar.
- Se crean separando con comas sus elementos. También se pueden poner entre paréntesis. Dos paréntesis sin nada en su interior es la tupla vacía.
- La función `tuple()` funciona igual que `list()`, toma como argumento una secuencia y lo convierte en una tupla. Si el argumento ya es una tupla, se devuelve sin cambios.
- El acceso se realiza con las mismas reglas que en las listas

## Ejemplo

- Cree una tupla con comas y paréntesis
- Cree una tupla vacía y una tupla de un solo elemento
- Convierta una lista y un string en una tupla
- Utilice el operador `[]` para acceder a una tupla



```
In [148]: 1,2,3
Out[148]: (1, 2, 3)

In [149]: (1,2,3)
Out[149]: (1, 2, 3)

In [150]: ()
Out[150]: ()

In [151]: 42,
Out[151]: (42,)

In [152]: tuple([1, 2, 3])
Out[152]: (1, 2, 3)

In [153]: tuple('abc')
Out[153]: ('a', 'b', 'c')

In [154]: tuple((1, 2, 3))
Out[154]: (1, 2, 3)

In [155]: x = 1,2,3
In [156]: x[1]
Out[156]: 2

In [157]: x[0:2]
Out[157]: (1, 2)

In [158]: _
```

# OPERACIONES

## Acceso y formateo

- Las operaciones de secuencias (indexación, división, multiplicación, pertenencia, longitud, mínimo y máximo) funcionan con strings. Sin embargo, las asignaciones de elementos o slices no funcionan porque los strings son inmutables.
- La solución inicial para formateo de strings era %, con un comportamiento similar a la función `printf` de C. A la izquierda del %, se coloca el string y a la derecha el valor a formatear. Los %s son especificadores de conversión. Marcan donde se insertan los valores. La s significa que se formatean como strings; si no lo son se convierten. Hay otros especificadores, ej.: % .3f formatea como un real con tres decimales. Para aplicar el formato se ejecuta `<string> % <valores>`
- Otra solución son las llamadas plantillas de strings con \$. Los argumentos con \$ son los argumentos de palabra clave. Para aplicar el formato se llama al método  
`<string>.substitute(<arg1>=<valor1>, ...)`

## Ejemplo

1. Cree un string con especificadores de conversión que imprima "Hola a todos. ¿Estais en la oficina ya?" si le pasamos los strings `todos` y `oficina`.
2. Repita el ejercicio con una plantilla de string y páselle otros strings.

# OPERACIONES (SOLUCIÓN)

## Acceso y formateo

- Las operaciones de secuencias (indexación, división, multiplicación, pertenencia, longitud, mínimo y máximo) funcionan con strings. Sin embargo, las asignaciones de elementos o slices no funcionan porque los strings son inmutables.
- La solución inicial para formateo de strings era %, con un comportamiento similar a la función `printf` de C. A la izquierda del %, se coloca el string y a la derecha el valor a formatear. Los %s son especificadores de conversión. Marcan donde se insertan los valores. La s significa que se formatean como strings; si no lo son se convierten. Hay otros especificadores, ej.: % .3f formatea como un real con tres decimales. Para aplicar el formato se ejecuta `<string> % <valores>`
- Otra solución son las llamadas plantillas de strings con \$. Los argumentos con \$ son los argumentos de palabra clave. Para aplicar el formato se llama al método `<string>.substitute(<arg1>=<valor1>, ...)`

```

HAL9000 — IPython: Users\HAL9000 — ipython — 67x18
In [17]: format = "Hola a %. ¿Estais en la % ya?"
In [18]: values = {'todos': 'oficina'}
In [19]: format % values
Out[19]: 'Hola a todos. ¿Estais en la oficina ya?'
In [20]: from string import Template
In [21]: tmpl = Template('Hola a $who. ¿Estais en la $where ya?')
In [22]: tmpl.substitute(who='todos', where='oficina')
Out[22]: 'Hola a todos. ¿Estais en la oficina ya?'
In [23]: tmpl.substitute(who='Javier y Eva', where='salida')
Out[23]: 'Hola a Javier y Eva. ¿Estais en la salida ya?'
In [24]:

```

## Ejemplo

- Cree un string con especificadores de conversión que imprima "Hola a todos. ¿Estais en la oficina ya?" si le pasamos los strings `todos` y `oficina`.
- Repita el ejercicio con una plantilla de string y páselle otros strings.

# USO DE F-STRINGS

- Las f-strings, o cadenas formateadas, se introdujeron a partir de Python 3.6.
- Permiten incluir expresiones dentro de cadenas literales utilizando llaves {}.
- Son una forma concisa y legible de formatear cadenas.
- Se pueden realizar cálculos y llamadas a funciones directamente dentro de la cadena de caracteres.

```
[>>> who = 'todos'
>>> where = 'oficina'
>>> f'Hola a {who}. Estais en la {where} ya?'
'Hola a todos. Estais en la oficina ya?'
>>> _
```

```
[>>>
>>> minutes = 20
>>> f'{minutes} minutos son {minutes*60} segundos'
'20 minutos son 1200 segundos'
>>> _
```

# MÉTODOS DE STRINGS

## Métodos

- `<string>.center(<value>, ['filler'])` centra el string rellenando ambos lados con un carácter, por defecto espacios
- `<string>.find('substring', [<start>, <end>])` devuelve el índice más a la izquierda del substring. Si no se encuentra, devuelve -1.
- `<string1>.join(<string2>)` une elementos de una secuencia
- `<string>.split(<separador>)` separa los elementos de un string en una lista.
- `<string>.lower()` convierte en minúsculas todo el string
- `<string1>.startswith(<string2>)` True si `string1` empieza con `string2`
- `<string1>.endswith(<string2>)` True si `string1` acaba con `string2`
- `<string1>.replace(string2, string3)` devuelve un string donde ha sustituido todas las ocurrencias de `string2` por `string2`
- `<string>.strip()` elimina los espacios en blanco de los extremos del string
- Para visualizar los métodos built-in se usa la función `dir(<objeto>)`

# MÉTODOS DE STRINGS (EJEMPLOS)

```

script3.py > ...
33 # (8) Dividir el texto en palabras
34 words = text.split()
35 print(words)
36
37 # (9) Unir las palabras con un guión
38 joined_text = "-".join(words)
39 print(joined_text)
40
41 # (10) Verificar si el texto original empieza con 'Este'
42 starts_with_Este = text.startswith('Este')
43 print(f"El texto empieza con 'Este': {starts_with_Este}")
44
45 # (11) Verificar si el texto original termina con 'string'
46 ends_with_string = text.endswith('string')
47 print(f"El texto termina con 'string': {ends_with_string}")
48
49 # (12) Eliminar espacios en blanco al inicio y al final del texto (si hay)
50 trimmed_text = text.strip()
51 print(f"Texto sin espacios en blanco al inicio y al final: '{trimmed_text}'")
52
53 # (13) Contar el número de palabras en el texto
54 num_words = len(words)
55 print(f"Número de palabras en el texto: {num_words}")
56
57 # (14) Reemplazar todos los espacios por un punto en el texto original
58 text_with_dots = text.replace(' ', '.')
59 print(text_with_dots)
60
61 # (15) Extraer una subcadena del texto original (de la posición 5 a la 15)
62 substring = text[5:15]
63 print(f"Subcadena de la posición 5 a la 15: '{substring}'")
64

```

```

script3.py > ...
1 text = 'Este es un texto largo para que hagamos unos ejercicios \
2 con los métodos del tipo de datos string'
3
4 # (1) Reemplazar 'hagamos' por 'hagais'
5 text = text.replace('hagamos', 'hagais')
6
7 # (2) Reemplace todas las 'e' por un '1' salvo la
8 # primera en una copia del string
9 idx = text.find('e')
10 text_copy = text[:idx+1] + text[idx+1:].replace('e', '1')
11 print(text_copy)
12
13 # (3) Convertir todo el texto a mayúsculas
14 text_upper = text.upper()
15 print(text_upper)
16
17 # (4) Convertir todo el texto a minúsculas
18 text_lower = text.lower()
19 print(text_lower)
20
21 # (5) Capitalizar la primera letra de cada palabra
22 text_title = text.title()
23 print(text_title)
24
25 # (6) Contar el número de apariciones de 'e' en el texto original
26 count_e = text.count('e')
27 print(f"Número de 'e' en el texto original: {count_e}")
28
29 # (7) Encontrar la posición de la primera aparición de 'texto'
30 pos_texto = text.find('texto')
31 print(f"Primera posición de 'texto': {pos_texto}")
32

```

# MAPPING

## Concepto

- En los diccionarios se hace referencia y se actualiza cada valor por un nombre. Los diccionarios consisten en pares de claves y sus valores sin orden establecido por lo que no se recorren.
- La sintaxis para crear un diccionario es `{<key1>:<value1>}`. Las claves son únicas en el diccionario, mientras que los valores pueden no serlo
- Un diccionario vacío (sin ningún elemento) se escribe con solo dos llaves `{}`. Con la función `dict()` se puede construir diccionarios a partir de secuencias. Para ello hay que pasarle parejas de claves-valor.

## Ejemplo

1. Cree un diccionario que contenga los siguientes nombres y teléfonos  
`'Alicia': '2341', 'Berta': '9102', 'Cecilio': '3258'`
2. Acceda al teléfono de de Alicia y actualícelo a `5678`
3. Cree un diccionario a partir de una lista con la función `dict()`.

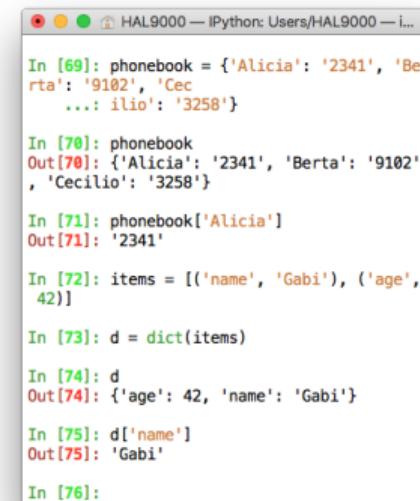
# MAPPING (SOLUCIÓN)

## Concepto

- En los diccionarios se hace referencia y se actualiza cada valor por un nombre. Los diccionarios consisten en pares de claves y sus valores sin orden establecido por lo que no se recorren.
- La sintaxis para crear un diccionario es `{<key1>:<value1>}`. Las claves son únicas en el diccionario, mientras que los valores pueden no serlo
- Un diccionario vacío (sin ningún elemento) se escribe con solo dos llaves `{}`. Con la función `dict()` se puede construir diccionarios a partir de secuencias. Para ello hay que pasarle parejas de claves-valor.

## Ejemplo

1. Cree un diccionario que contenga los siguientes nombres y teléfonos  
`'Alicia': '2341', 'Berta': '9102', 'Cecilio': '3258'`
2. Acceda al teléfono de de Alicia y actualícelo a **5678**
3. Cree un diccionario a partir de una lista con la función `dict()`.



```
In [69]: phonebook = {'Alicia': '2341', 'Be
rta': '9102', 'Cec
...: ilio': '3258'}
In [70]: phonebook
Out[70]: {'Alicia': '2341', 'Berta': '9102',
'Cecilio': '3258'}
In [71]: phonebook['Alicia']
Out[71]: '2341'
In [72]: items = [('name', 'Gabi'), ('age',
42)]
In [73]: d = dict(items)
In [74]: d
Out[74]: {'age': 42, 'name': 'Gabi'}
In [75]: d['name']
Out[75]: 'Gabi'
In [76]:
```

# MAPPING

## Concepto

- La función `del <dict>[<key>]` elimina una pareja clave-valor
- El método `<dict>.clear()` elimina todas las entradas pero no el diccionario.
- El método `<dict>.copy()` crea una copia del diccionario no enlazada.
- El método `<dict>.fromkeys()` crea una copia del diccionario aunque solo con las claves.
- Los métodos `<dict>.keys()` y `<dict>.values()` devuelven una vista de las claves y los valores. Se pueden convertir en una lista con `list()`.
- El método `<dict>.items()` crea una copia del diccionario no enlazada.

## Ejemplo

1. Cree un diccionario
2. Elimine una entrada y posteriormente todas sus entradas

```
HAL9000 — IPython: Users/HAL9000 — python —
In [78]: purse = dict()
In [79]: purse['money']=12
In [80]: purse['candy']=3
In [81]: purse['tissues']=75
In [82]: purse
Out[82]: {'candy': 3, 'money': 12, 'tissues': 75}
In [83]: purse['candy']=purse['candy']+3
In [84]: purse
Out[84]: {'candy': 6, 'money': 12, 'tissues': 75}
In [85]: del purse['candy']
In [86]: purse
Out[86]: {'money': 12, 'tissues': 75}
In [87]: purse2 = purse
In [88]: purse2.clear()
In [89]: purse2
Out[89]: {}
In [90]: purse
Out[90]: {}
In [91]:
```

# RESUMEN

## Funciones de listas y tuplas

Función	Descripción
abs(number)	Devuelve el valor absoluto de un número
len(seq)	Devuelve la longitud de una secuencia
list(seq)	Convierte una secuencia a lista
max(args)	Devuelve el máximo de una secuencia o conjunto de argumentos
min(args)	Devuelve el mínimo de una secuencia o conjunto de argumentos
sorted(seq)	Devuelve una lista ordenada

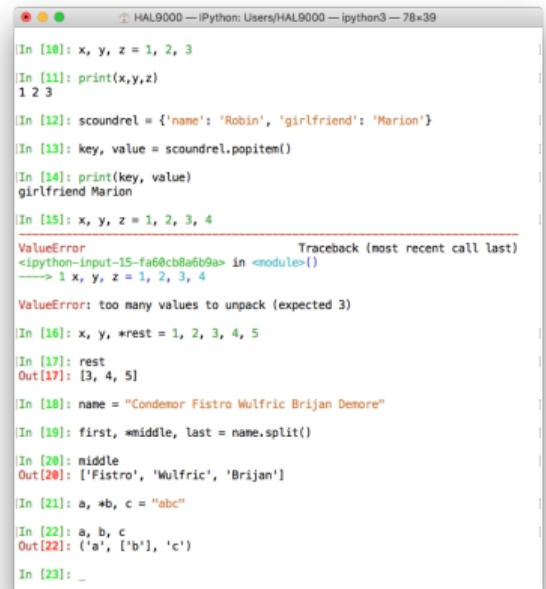
# DESEMPAQUETADO DE SECUENCIAS

## Creación de listas

- Puede realizar varias asignaciones diferentes simultáneamente. Para ello la salida de LHS tiene que ser del mismo número del RHS  
`var1, var2 = <fun returning two values>`
- Útil cuando una función o método devuelve una tupla (u otra secuencia u objeto iterable). Si se desea recuperar (y eliminar) un par <key-value> arbitrario de un diccionario, se puede usar `popitem`, y luego desacoplar la tupla devuelta en dos variables.
- En lugar de asegurar que el número de valores coincide exactamente, puede reunir los restantes con el operador \*
- RHS puede ser cualquier clase de secuencia, pero la variable con asterisco siempre completará el contenido de la lista

## Ejemplo

- Cree una tupla de 6 elementos y asigne los dos primeros y el penúltimo a variables, el resto a listas.



```

In [10]: x, y, z = 1, 2, 3
In [11]: print(x,y,z)
1 2 3

In [12]: scoundrel = {'name': 'Robin', 'girlfriend': 'Marion'}
In [13]: key, value = scoundrel.popitem()
In [14]: print(key, value)
girlfriend Marion

In [15]: x, y, z = 1, 2, 3, 4
ValueError: too many values to unpack (expected 3)
<ipython-input-15-fa6@cb8a6b9a> in <module>()
      1 x, y, z = 1, 2, 3, 4
      2
      3     ^
      4 Traceback (most recent call last)

ValueError: too many values to unpack (expected 3)
In [16]: x, y, *rest = 1, 2, 3, 4, 5
In [17]: rest
Out[17]: [3, 4, 5]

In [18]: name = "Condemor Fistro Wulfric Brijan Demore"
In [19]: first, *middle, last = name.split()
In [20]: middle
Out[20]: ['Fistro', 'Wulfric', 'Brijan']

In [21]: a, *b, c = "abc"
In [22]: a, b, c
Out[22]: ('a', ['b'], 'c')

In [23]: _

```

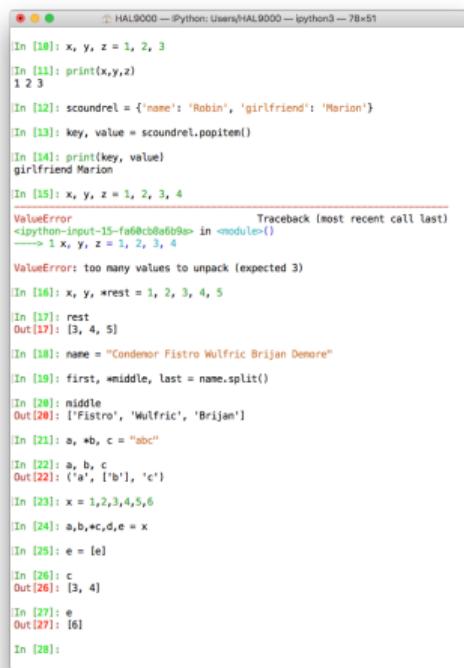
# DESEMPAQUETADO DE SECUENCIAS (SOLUCIÓN)

## Creación de listas

- Puede realizar varias asignaciones diferentes simultáneamente. Para ello la salida de LHS tiene que ser del mismo número del RHS  
`var1, var2 = <fun returning two values>`
- Útil cuando una función o método devuelve una tupla (u otra secuencia u objeto iterable). Si se desea recuperar (y eliminar) un par <key-value> arbitrario de un diccionario, se puede usar `popitem`, y luego desacoplar la tupla devuelta en dos variables.
- En lugar de asegurar que el número de valores coincide exactamente, puede reunir los restantes con el operador \*
- RHS puede ser cualquier clase de secuencia, pero la variable con asterisco siempre completará el contenido de la lista

## Ejemplo

- Cree una tupla de 6 elementos y asigne los dos primeros y el penúltimo a variables, el resto a listas.



```

In [10]: x, y, z = 1, 2, 3
In [11]: print(x,y,z)
1 2 3

In [12]: scoundrel = {'name': 'Robin', 'girlfriend': 'Marion'}
In [13]: key, value = scoundrel.popitem()
In [14]: print(key, value)
girlfriend Marion

In [15]: x, y, z = 1, 2, 3, 4
ValueError: too many values to unpack (expected 3)
<ipython-input-15-fa60cb8a6b9e> in <module>()
----> 1 x, y, z = 1, 2, 3, 4
ValueError: too many values to unpack (expected 3)

In [16]: x, y, *rest = 1, 2, 3, 4, 5
In [17]: rest
Out[17]: [3, 4, 5]

In [18]: name = "Condenor Fistro Wulfric Brijan Demore"
In [19]: first, *middle, last = name.split()
In [20]: middle
Out[20]: ['Fistro', 'Wulfric', 'Brijan']

In [21]: a, *b, c = "abc"
In [22]: a, b, c
Out[22]: ('a', 'b', 'c')

In [23]: x = 1,2,3,4,5,6
In [24]: a,b,*c,d,e = x
In [25]: e = [e]
In [26]: c
Out[26]: [3, 4]

In [27]: e
Out[27]: [6]

In [28]:

```

# ASIGNACIONES

## Asignaciones encadenadas

- Las asignaciones encadenadas son un acceso directo cuando desea vincular varias variables al mismo valor `x = y = somefunction()` que es lo mismo que

```
y = somefunction()  
x = y
```

## Asignaciones aumentadas

- En lugar de escribir `x = x + 1`, puede poner el operador de expresión (en este caso `+`) antes del operador de asignación (`=`) y escribir `x += 1`. Esto funciona con los estándares operadores, como `*`, `/`, `%`, y con otros tipos de datos como strings.

## Ejemplo

- Realice una asignación encadenada
- Realice asignaciones aumentadas

```
HAL9000 — IPython...  
In [28]: x = y = 3*4  
In [29]: x  
Out[29]: 12  
In [30]: y  
Out[30]: 12  
In [31]: fnord = 'foo'  
In [32]: fnord += 'bar'  
In [33]: fnord *= 2  
In [34]: fnord  
Out[34]: 'foobarfoobar'  
In [35]: _
```

## 1. Entorno

## 2. Fundamentos

## 3. Colecciones

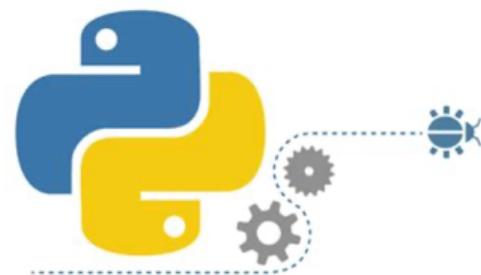
## 4. Control de flujo

### 4.1 Condicionales

### 4.2 Loops

### 4.3 List comprehension

## 5. Funciones



# ASIGNACIONES Y COMPARACIÓN

## Operadores de comparación

- Internamente **True==1** y **False==0**. **bool()** devuelve un valor lógico según el argumento que se le pase

Op.	Descripción	Op.	Descripción
<	Menor que	>	Mayor que
<=	Menor o igual que	<=	Mayor o igual que
==	Igual que	!=	Distinto de
x is y	x es el mismo objeto que y	x not is y	x no es el mismo objeto que y
x in y	x es miembro del container y	x not in y	x no es miembro del container y
and	AND Lógico	or	OR Lógico

- Se trata de expresiones booleanas que devuelven **True** o **False** y controlan el flujo del script. Los operadores de comparación acceden a las variables pero no cambian los resultados.
  - La indentación determina el alcance de la ejecución

```
[In [4]: 'Smaller' < 'smaller'
Out[4]: True

[In [5]: True + False + 50
Out[5]: 51

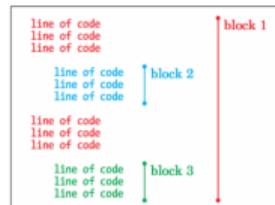
[In [6]: x = y = [1,2,3]
[In [7]: z = [1,2,3]

[In [8]: x == z
Out[8]: True

[In [9]: x is z
Out[9]: False

[In [10]: y[0] = 6
[In [11]: x
Out[11]: [6, 2, 3]

In [12]: _
```



## Ejemplo

- Compare dos strings idénticos pero en una la primera en mayúsculas

## SENTENCIA IF | IF ELSE

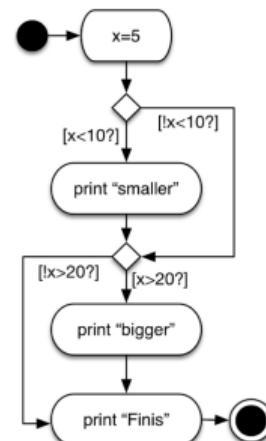
if

- Se emplea la sentencia **if** seguida de una expresión lógica y **:**. Se incrementa la indentación detrás de cada sentencia condicional. Se mantiene la indentación para indicar el alcance del bloque. Sustituyen a **{}** e indica las líneas afectadas por **if**. Se reduce la indentación para indicar el final del bloque. Las líneas en blanco y los comentarios se ignoran.

- En caso que la condición lógica no se cumpla se realizan acciones alternativas con **else**

```
if <Boolean expression>:  
    <some indented code>  
else: # Notice the colon here too  
    <other indented code>
```

- | a sintaxis de las expresión condicionales `if <cond> else <code>`



## Ejemplo

- Cree un script que pida por consola ingresos brutos anuales. Si igualan o superan los 60.000 imprimirá por pantalla **Prepárate para pagar a Hacienda**, en caso contrario dirá **Vas a pagar de todas formas**
  - Pregunte nombre y apellido. Si el apellido es **Gomez** responda **Bienvenido a su casa**, mientras que si es otro responda **¿y Vd. quien es?**
  - Reescríbalo con una expresión condicional

The screenshot shows a Jupyter Notebook environment. The left sidebar lists files: Desktop, .DS\_Store, localized, AAC\_Gantt.ipynb, Captura de pantalla, and figures.png. The main area displays a code cell with the following Python script:

```
untitled
1 x= 5
2 if x < 10:
3     print 'Smaller'
4
5 if x > 20:
6     print 'Bigger'
7
8 print 'Finis'
```

The status bar at the bottom indicates the file is 'untitled', has 8 lines, and is in Python mode.

# SENTENCIA IF | IF ELSE (SOLUCIÓN)

## if

- Se emplea la sentencia `if` seguida de una expresión lógica y `:`. Se incrementa la indentación detrás de cada sentencia condicional. Se mantiene la indentación para indicar el alcance del bloque. Sustituyen a `{}` e indica las líneas afectadas por `if`. Se reduce la indentación para indicar el final del bloque. Las líneas en blanco y los comentarios se ignoran.
- En caso que la condición lógica no se cumpla se realizan acciones alternativas con `else`

```
if <Boolean expression>:  
    <some indented code>  
else: # Notice the colon here too  
    <other indented code>
```

- La sintaxis de las expresión condicionales `<code> if <cond> else <code>`.

## Ejemplo

- Cree un script que pida por consola ingresos brutos anuales. Si igualan o superan los 60.000 imprimirá por pantalla **Prepárate para pagar a Hacienda**, en caso contrario dirá **Vas a pagar de todas formas**
- Pregunte nombre y apellido. Si el apellido es **Gomez** responda **Bienvenido a su casa**, mientras que si es otro responda **¿y Vd. quien es?**
- Reescríbalo con una expresión condicional

## Consejo

- Defina celdas para ejecuciones parciales de un script

```
6 #author: HAL9000 | José Luis Salmerón|  
7 %% Definición de celdas  
8  
9 # (1) Cree un script que pida por consola  
10 # los ingresos brutos anuales  
11 # Ingresos = input('Podría decirme cuanto gana?')  
12  
13 # Resultado  
14 if int(ingresos) >= 60000:  
15     print('Prepárate para pagar a Hacienda!')  
16 else:  
17     print('Vas a pagar de todas formas!')  
18  
19 #%%  
20  
21 # (2) Pregunta nombre y apellido y responde en función  
22 # del apellido  
23  
24 nombre = input('¿Cuál es su nombre y apellido? ')  
25  
26 if nombre.endswith('Gómez'):  
27     print('Bienvenido a su casa')  
28 else:  
29     print('¿y Vd. quien es?')  
30  
31 #%%  
32  
33  
34 # (3) Con expresión condicional  
35  
36 nombre = input('¿Cuál es su nombre y apellido? ')  
37  
38 print('Bienvenido') if nombre.endswith('Gómez') else print('¿y Vd. quien es?')
```

# SENTENCIA IF ELIF

## Aserciones

- A veces se prefiere que la ejecución falle si no se cumple una condición que en un momento posterior. Se puede exigir que ciertas condiciones sean ciertas (por ejemplo, al verificar las propiedades requeridas de los parámetros de sus funciones o como ayuda durante las pruebas iniciales y la depuración).
- Para ello la declaración es `assert <condicion>`

## Condiciones múltiples y anidadas

- Para verificar varias condiciones se usa `elif`

```
if <Boolean expression>:  
    <some indented code>  
elif <Boolean expression>:  
    <some indented code>  
else:  
    <other indented code>
```

The screenshot shows an IPython notebook interface with the following content:

```
[In [59]: age = 10  
[In [60]: assert 0 < age < 100  
[In [61]: age = 120  
[In [62]: assert 0 < age < 100  
AssertionErrorTraceback (most recent call last)  
<ipython-input-62-5f6a336febcc> in <module>()  
----> 1 assert 0 < age < 100  
AssertionError:  
In [63]:
```

The notebook shows four input cells (In [59], In [60], In [61], In [62]) and one output cell (AssertionError). The error occurs at In [62] because the variable 'age' is assigned the value 120, which does not satisfy the condition 0 < age < 100.

# EJERCICIO CONDICIONALES

## Enunciado

- Escriba un script que mediante entradas por consola:
  - Calcula el precio que se cobra a un cliente según las horas de trabajo, introduciendo número de horas y precio por hora.
  - Reescriba el script anterior, en el caso de que conceda una reducción del 10% en el precio por hora por más de 100 horas de trabajo. La respuesta por consola ha de tener dos decimales.
  - Reescriba el script anterior, en el caso que los clientes de AA.PP. paguen de IVA solo el 6% y los privados el 12%. La respuesta por consola ha de tener dos decimales. Tenga en cuenta que las administraciones públicas se pueden llamar Administraciones Públicas, AA.PP., o administración en cualquier capitalización.

# EJERCICIO CONDICIONALES (SOLUCIÓN)

```
script4.py > ...
1 # Solicitar los datos al usuario
2 horas = float(input("Introduzca el número de horas: "))
3 precio_por_hora = float(input("Introduzca el precio por hora: "))
4 tipo_cliente = input("Introduzca el tipo de cliente (Administraciones \ 
5 Pùblicas, AA.PP., administración o privado): ")
6
7 # Calcular el precio total sin descuento
8 precio_total = horas * precio_por_hora
9
10 # Aplicar el descuento del 10% si las horas son más de 100
11 if horas > 100:
12     precio_por_hora_con_descuento = precio_por_hora * 0.90
13     precio_total = horas * precio_por_hora_con_descuento
14
15 # Calcular el IVA según el tipo de cliente
16 if tipo_cliente.lower() in ["administraciones pùblicas", "aa.pp.", "administración"]:
17     iva = 0.06
18 else:
19     iva = 0.12
20
21 # Calcular el precio total con IVA
22 precio_total_con_iva = precio_total * (1 + iva)
23
24 # Mostrar el precio final
25 print(f"El precio final a cobrar al cliente es: {precio_total_con_iva:.2f} euros")
26
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Introduzca el número de horas: 20  
Introduzca el precio por hora: 35  
Introduzca el tipo de cliente (Administraciones Pùblicas, AA.PP., administración o privado): privado  
El precio final a cobrar al cliente es: 784.00 euros

Ln 5, Col 1 Spaces: 4 UTF-8 LF ↵ Python 3.12.4 64-bit

# BUCLES

## while

- Para loops se usa **where**  

```
while <Boolean expression>:  
    <some indented code>
```
- Para salir del loop es necesario que la condición sea **False** en algún momento.

## Ejemplo

- Escriba un script que pida su nombre por consola y que siga pidiéndolo hasta que lo introduzca. Luego salúdele.
- Una vez introducido, compruebe que ha introducido al menos un nombre y apellidos, si no es así pídaselo otra vez. Salude por el primer apellido.
- Reescriba el snippet anterior para que si el primer nombre acaba en a el saludo **Sra.** <apellido1>, en caso contrario **Sr.** <apellido1>
- Reescriba el snippet anterior para que además compruebe que uno de los dos apellidos pertenecen a una serie de apellidos autorizados.
- Reescriba el snippet anterior para que además compruebe que los dos apellidos están autorizados.  
Tip: Tenga en cuenta las distintas capitalizaciones.

# BUCLES (SOLUCIÓN)

## while

- Para loops se usa where
  - while <Boolean expression>:  
    <some indented code>
- Para salir del loop es necesario que la condición sea **False** en algún momento.

## Ejemplo

- Escriba un script que pida su nombre por consola y que siga pidiéndolo hasta que lo introduzca. Luego salúdele.
- Una vez introducido, compruebe que ha introducido al menos un nombre y apellidos, si no es así pídaselo otra vez. Salude por el primer apellido.
- Reescriba el snippet anterior para que si el primer nombre acaba en a el saludo **Sra.** <apellido1>, en caso contrario **Sr.** <apellido1>
- Reescriba el snippet anterior para que además compruebe que uno de los dos apellidos pertenezcan a una serie de apellidos autorizados.
- Reescriba el snippet anterior para que además compruebe que los dos apellidos están autorizados.

```
script5.py ...
1 # Etapa 1: Solicitar el nombre y asegurarse de que se introduzca algo
2 nombre = ''
3 while nombre=='':
4     nombre = input('Por favor, introduzca su nombre: ')
5
6 # Etapa 2: Verificar que se introduzca al menos un nombre y un apellido
7 while len(nombre.split(' '))<=1:
8     nombre = input('Por favor, introduzca su nombre y apellidos: ').strip()
9
10 # Saludar por el primer apellido
11 print(f'Hola, {nombre.split(" ")[0]}!')
12
13 # Etapa 3: Saludar según el género del primer nombre
14 if nombre.split(' ')[0].lower().endswith('a'):
15     print('Sra. {nombre.split(" ")[0]}')
16 else:
17     print('Sr. {nombre.split(" ")[0]}')
18
19 # Etapa 4: Comprobar si los apellidos están en una lista de apellidos autorizados
20 apellidos_autorizados = ['Gómez', 'López', 'Martínez', 'García']
21
22 # Verificar que al menos uno de los apellidos esté en la lista de autorizados
23 if nombre.split(' ')[1] in apellidos_autorizados or \
24     nombre.split(' ')[1:-1] in apellidos_autorizados:
25     print(f'Bienvenido, {nombre}!')
26 else:
27     print('Lo siento, sus apellidos no están autorizados.')
28
29 # Etapa 5: Comprobar que ambos apellidos están en la lista de apellidos autorizados
30 if nombre.split(' ')[1] in apellidos_autorizados and \
31     nombre.split(' ')[1:-1] in apellidos_autorizados:
32     print(f'Bienvenido, {nombre}!')
33 else:
34     print('Lo siento, ambos apellidos deben estar autorizados para ingresar.')
```

# BUCLES

## for

- Para iterar sobre una colección de objetos se usa **for**

```
for <element> in <collection>:  
    <some indented code>
```

- Se sale del loop cuando finaliza el recorrido. Si queremos recorrer una lista de número podemos usar la función built-in **range([start=0], stop, [step=1])**.

- Para la iteración paralela está la función built-in **zip**, que comprime secuencias y devuelve tuplas. El valor de retorno es un objeto **zip**, destinado a la iteración, pero se puede convertir utilizando la lista para ver sus contenidos.

- Para salir de un bucle use **break**, **continue** termina la iteración y salta a la siguiente y para que no pase nada en un bucle no terminado **pass**.

- La función **zip** funciona con tantas secuencias como desee. Cuando las secuencias son de diferentes longitudes se detiene cuando termina la secuencia más corta

```
Last login: Sun Nov 12 23:28:09 on ttys000
You have new mail.
MacBook-Pro-de-Computational:- HAL9000:~ ipython3
Python 3.6.3 |Anaconda, Inc.| (default, Oct  6 , 2016, 12:04:38)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type "?" for help.

In [1]: names = ['anne', 'beth', 'george', 'damon']
...: ages = [12, 45, 32, 102]
...:

In [2]: list(zip(names, ages))
Out[2]: [('anne', 12), ('beth', 45), ('george', 32), ('damon', 102)]

In [3]:
```

## Ejemplo

- Escriba un programa para construir el siguiente patrón



# BUCLES (SOLUCIÓN)

## for

- Para iterar sobre una colección de objetos se usa **for**

```
for <element> in <collection>:  
    <some indented code>
```
- Se sale del loop cuando finaliza el recorrido. Si queremos recorrer una lista de número podemos usar la función built-in **range([start=0], stop, [step=1])**.
- Para la iteración paralela está la función built-in **zip**, que comprime secuencias y devuelve tuplas. El valor de retorno es un objeto **zip**, destinado a la iteración, pero se puede convertir utilizando la lista para ver sus contenidos.
- Para salir de un bucle use **break**, **continue** termina la iteración y salta a la siguiente y para que no pase nada en un bucle no terminado **pass**.
- La función **zip** funciona con tantas secuencias como desee. Cuando las secuencias son de diferentes longitudes se detiene cuando termina la secuencia más corta

```
6 @author: HAL9000  
7 """  
8  
9 # If using Python 2.7  
10 # from __future__ import print_function  
11  
12 n=5;  
13 for i in range(n):  
14     for j in range(i):  
15         # El argumento end sustituye \n al final  
16         print ('*', end='')  
17     print()  
18  
19 for i in range(n,0,-1):  
20     for j in range(i):  
21         print('*', end='')  
22     print()
```

## Ejemplo

- Escriba un programa para construir el siguiente patrón



# EJERCICIOS BUCLES

## Enunciados

- Escriba un script para contar la cantidad de números pares e impares de una serie de números.
- Escriba un script que imprima cada elemento y su tipo correspondiente de la siguiente lista `datalist = [1452, 11.23, 1+2j, True, 'w3resource', (0, -1), [5, 12], {'class':'V', 'section':'A'}]`
- Escriba un script que tome dos dígitos m (fila) y n (columna) como entrada y genere una matriz bidimensional. El valor del elemento en la i-ésima fila y la j-ésima columna de la matriz debe ser `i * j`.
- Escriba un script para encontrar números entre 100 y 400 (ambos incluidos) donde cada dígito sea un número par. Los números obtenidos deben imprimirse en una secuencia separada por comas.
- Escriba un script para concatenar los siguientes diccionarios para crear uno nuevo:  
`dic1={1:10, 2:20}, dic2={3:30, 4:40}, dic3={5:50,6:60}`. Resultado: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

# EJERCICIOS BUCLES (SOLUCIÓN)

```

7 #author: HAL9000 | José Luis Salmerón
8 """
9
10 #%% (1) Número de pares e impares
11
12 numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9) # Tupla
13 count_odd = 0
14 count_even = 0
15 for x in numbers:
16     if not x % 2:
17         count_even+=1
18     else:
19         count_odd+=1
20
21 print("Number of even numbers :",count_even)
22 print("Number of odd numbers :",count_odd)
23
24 #%% (2) Imprime los elementos y tipo de la lista
25
26 datalist = [1452, 11.23, 1+2j, True, 'w3resource', (0, -1), [5, 12],
27 {"class":'V', "section":'A'}]
28 for item in datalist:
29     print ("Type of ",item, " is ", type(item))
30
31 #%% (3) Toma m (fila) y n (columna) y genere una matriz bidimensional.
32 # El valor cada elemento es i * j.
33
34 row_num = int(input("Input number of rows: "))
35 col_num = int(input("Input number of columns: "))
36 multi_list = [[0 for col in range(col_num)] for row in range(row_num)]
37
38 for row in range(row_num):
39     for col in range(col_num):
40         multi_list[row][col]= row*col
41
42 print(multi_list)
43
44 #%% (4) Dígitos pares
45
46 items = []
47 for i in range(100, 401):
48     s = str(i)
49     if (int(s[0])%2==0) and (int(s[1])%2==0) and (int(s[2])%2==0):
50         items.append(s)
51 print( ",".join(items))
52
53 #%% (5) Concatenación de diccionarios
54
55 dic1={1:10, 2:20}
56 dic2={3:30, 4:40}
57 dic3={5:50, 6:60}
58 dic4 = {}
59 for d in (dic1, dic2, dic3):
60     dic4.update(d)
61 print(dic4)

```

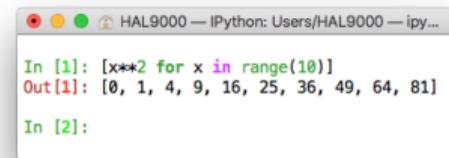
# LISTAS DE OTRAS LISTAS

[...]

- La comprensión de listas es una forma de hacer listas de otras listas.
- Funciona de manera similar a los bucles `for <output> for <element> in <lista> if <condition>` (if opcional)

## Ejemplo

- Escriba un script para crear una lista concatenando otra lista cuyo rango vaya de 1 a n.  
Lista `my_list = ['p', 'q']`  
Resultado: `['p1', 'q1', 'p2', 'q2', 'p3', 'q3', 'p4', 'q4']`
- Tome la primera letra de cada palabra `['this', 'is', 'a', 'list', 'of', 'words']` y haga una lista
- Reescriba con list comprehension:  
`for x in range(10):  
 squares.append(x**2)`
- A partir del string `'The quick brown fox jumps over the lazy dog'` cree una lista donde cada uno de los elementos es otra lista con cada palabra en mayúsculas, minúsculas y el número de caracteres.



HAL9000 — IPython: Users/HAL9000 — ipy...

```
In [1]: [x**2 for x in range(10)]
Out[1]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

In [2]:
```

# LISTAS DE OTRAS LISTAS (SOLUCIÓN)

[...]

- La comprensión de listas es una forma de hacer listas de otras listas.
- Funciona de manera similar a los bucles **for**  
**[<output> for <element> in <lista> if <condition>]** (if opcional)

## Ejemplo

- Cree una lista concatenando otra lista con rango 1 a n. Lista: `my_list = ['p', 'q']`  
 Resultado: `['p1', 'q1', 'p2', 'q2', 'p3', 'q3', 'p4', 'q4']`
- Tome la primera letra de cada palabra `['this','is','a','list','of','words']` y haga una lista
- Reescriba con list comprehension:  
`for x in range(10):  
 squares.append(x**2)`
- Del string `'The quick brown fox jumps over the lazy dog'` cree una lista donde cada uno de los elementos es otra lista con palabras mayúsculas, minúsculas y el número de caracteres.
- Extraiga las vocales con list comprehension del string  
`I love python and computer science`

```

6 @author: HAL9000 | José Luis Salmerón
7 ====
8
9 #%% (1) Concatenado de listas
10 my_list = ['p', 'q']
11 n = 4
12 new_list = ['{}{}'.format(x, y) for y in range(1, n+1) for x in my_list]
13 print(new_list)
14
15 #%% (2) Lista de primer carácter de strings
16 list_of_words = ['this','is','a','list','of','words']
17 items = [word[0] for word in list_of_words]
18 print(items)
19
20
21 #%% (3) Reescritura de for
22 squares = []
23 for x in range(10):
24     squares.append(x**2)
25
26 print(squares)
27
28 squares = [x**2 for x in range(10)]
29 print(squares)
30
31 #%% (4) 'The quick brown fox jumps over the lazy dog'
32 words = 'The quick brown fox jumps over the lazy dog'.split()
33 stuff = [(w.upper(), w.lower(), len(w)) for w in words]
34 print(stuff)
35
36 #%% (5) Extraiga las vocales de 'I love python and computer science'
37 vowels = [x for x in 'I love python and computer science'
38           if x.upper() in ['A','E','I','O','U']]
39 print(vowels)

```

# RESUMEN

## Funciones de listas y tuplas

Función	Descripción
<code>abs(number)</code>	Devuelve el valor absoluto de un número
<code>len(seq)</code>	Devuelve la longitud de una secuencia
<code>list(seq)</code>	Convierte una secuencia a lista
<code>max(args)</code>	Devuelve el máximo de una secuencia o conjunto de argumentos
<code>min(args)</code>	Devuelve el mínimo de una secuencia o conjunto de argumentos
<code>sorted(seq)</code>	Devuelve una lista ordenada
<code>chr(n)</code>	Devuelve un string de un carácter cuando se pasa el ordinal n ( $0 \leq n < 256$ )
<code>eval(source[, globals[, locals]])</code>	Evalúa una cadena como una expresión y devuelve el valor
<code>exec(source[, globals[, locals]])</code>	Evalúa y ejecuta una cadena como una declaración
<code>range([start,] stop[, step])</code>	Crea una lista de enteros
<code>reversed(seq)</code>	Devuelve la secuencia en orden inversa
<code>sorted(seq[, cmp][, key][, reverse])</code>	Ordena la secuencia
<code>xrange([start,] stop[, step])</code>	Crea un objeto xrange para iteración
<code>zip(seq1, seq2,...)</code>	Crea una nueva secuencia para iteraciones en paralelo

## 1. Entorno

## 2. Fundamentos

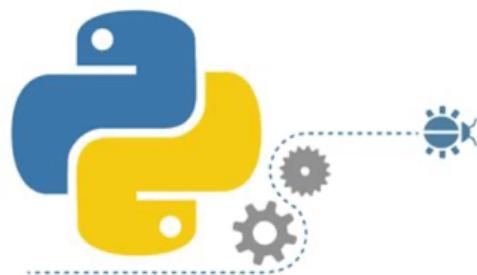
## 3. Colecciones

## 4. Control de flujo

## 5. Funciones

### 5.1 Invocación

### 5.2 Creación



# INVOCACIÓN

## Definición de funciones

- Una función es un objeto que se puede invocar (con/sin parámetros), que realiza una acción y puede devolver un objeto.
- Se puede saber si algo es invocable o no con la función built-in **callable**
- Para documentar se agregan comentarios con **#** o con strings después de **def**. Si coloca una cadena al comienzo de una función, se almacena como parte de la función y se denomina docstring.
- Se emplea la sentencia **def** con la siguiente sintaxis

```
def <nombre_función>(<args>):
    <indented code>
```

## Ejemplo

- Escriba una función que devuelva una lista de números de Fibonacci, pero en lugar de leer un número del usuario, lo recibe como parámetro.
- Acceda al **docstring** de la función creada

```
6 @author: HAL9000 | José Luis Salmerón
7 """
8 %% Objets invocables
9 import math
10
11 x = 1
12
13 y = math.sqrt
14
15 # Test
16 print( callable(x) )
17 print( callable(y) )
18
19
20 %% Función con la serie de Fibonacci
21
22 def fibs(num):
23     'Devuelve la serie de Fibonacci'
24     result = [0, 1]
25     for i in range(num-2):
26         result.append(result[-2] + result[-1])
27     return result
28
29
30 # Test
31 print( fibs(8) )
32 print( fibs(10) )
33
34 print( fibs.__doc__ )
35 help(fibs)
```

# CREACIÓN

## Tipos de parámetros

- Los argumentos anteriores se denominan posicionales.
- Parámetros opcionales: Se asigna al parámetro un tipo de dato en la cabecera de función.
- Número variable: Se sitúa \* como prefijo del parámetro que coloca todos los valores en la misma tupla.
- Al igual que las asignaciones, el parámetro con \* puede estar en cualquier posición. A diferencia de las asignaciones, debe especificar los parámetros finales por su nombre.
- El parámetro con \* no recoge los argumentos con valor por defecto, para ello hay que usar \*\* que los inserta en un diccionario en lugar de una tupla.

## Ejemplo

- Escriba una función que devuelva la potencia de un número.
- Escriba una función que imprima los parámetros que se le pasen.
  - Todos los parámetros.
  - Con los argumentos de número variables entre dos posicionales.
  - Que funcione con parámetros con valor por defecto.
  - Que combine todos los tipos de parámetros.

```

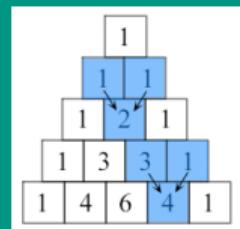
6 @author: HAL9000 | José Luis Salmerón
7
8 """
9 #%% Parámetrosopcionales
10 def my_pow(x, exp=2):
11     return x**exp
12
13
14 # test
15 print( my_pow(2) )
16 print( my_pow(2, 3) )
17
18 #%% Número indefinido deparámetros
19
20
21 def print_params(*params):
22     print(params)
23
24 # Test
25 print_params(1, 'nice')
26 print_params(1,2,3, 'nice')
27
28 def in_the_middle(x, *y, z):
29     print(x, y, z)
30
31 # Test
32 in_the_middle(1, 2, 3, 4, 5, z=7)
33
34 def print_params_2(title, *params):
35     print(title)
36     print(params)
37
38 # Test
39 print_params_2('Params:', 1, 2, 3)
40 print_params_2('Hm...', something=42) # Error
41
42 #%% Número indefinido deargumentos y con
43 # valor por defecto
44 def print_params_3(**params):
45     print(params)
46
47 # Test
48 print_params_3(x=1, y=2, z=3)
49 #print_params_3('Hola', x=1, y=2, z=3) # Error
50
51 def print_params_4(x, y, z=3, *pospar, **keypar):
52     print(x, y, z)
53     print(pospar)
54     print(keypar)
55
56 # Test
57 print_params_4(1, 2, 3, 5, 6, 7, foo=1, bar=2)
58 print_params_4(1, 2)

```

# EJERCICIO FUNCIONES

## Enunciado

- Escriba una función que multiplique todos los números de una lista
- Reescriba la función anterior para que multiplique todos los números pasados como parámetros
- Escriba una función que cambie el orden de un string
- Escriba una función que imprima las primeras  $n$  filas del triángulo de Pascal



- Escriba una función que concatene strings con un separador (por defecto -)

# EJERCICIO FUNCIONES (SOLUCIÓN)

```

5
6 @author: HAL9000 | José Luis Salmerón
7 """
8
9 #%% (1)
10
11 def list_mult(numbers):
12     'Multiplica todos los números de una lista'
13     total = 1
14     for x in numbers:
15         total *= x
16     return total
17
18 # Test
19 print(list_mult((5, 2, 3, -1, 7)))
20 print(list_mult((5, 2, 45, 7)))
21
22 #%% (2)
23
24 def list_mult2(*numbers):
25     'Multiplica los números pasados como argumentos'
26     total = 1
27     for x in numbers:
28         total *= x
29     return total
30
31 # Test
32 print(list_mult((5, 2, 3, -1, 7)))
33 print(list_mult((5, 2, 45, 7)))

```

```

5 #%% (3)
6
7 def reverse_string(str1):
8     'Cambia el orden de un string'
9     rstr1 = ''
10    index = len(str1)
11    while index > 0:
12        rstr1 += str1[ index - 1 ]
13        index = index - 1
14    return rstr1
15
16 # Test
17 print( reverse_string('1234abcd') )
18 print( reverse_string('ejaugnel narG') )
19
20 #%% (4)
21
22 def pascal_triangle(n):
23     'Imprime el triangulo de Pascal'
24     row = [1]
25     y = [0]
26     for x in range(max(n,0)):
27         print(row)
28         row=[l+r for l,r in zip(row+y, y+row)]
29
30 pascal_triangle(6)
31
32 #%% (5)
33
34 def join(*args, sep="-"):
35     'Concatena strings con un separador'
36     return sep.join(args)
37
38 # Test
39 print(join("Une", "estas", "palabras"))

```

# TEMA 0.2: INTRO A PYTHON

Asignatura: Inteligencia Artificial

3º Curso Ingeniería Informática

Fecha de actualización: 21st August 2024

Prof. Dr. José Luis Salmerón



Escuela Politécnica Superior  
**CUNEF Universidad**

