

Introducción a Python

09/09/20 09:25 CIB Research Group 1/9

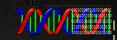




Juan Antonio Romero (aromero@uco.es)

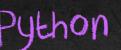
http://www.uco.es/-aromero

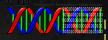
Departamento de Informática y Análisis Numérico



Python

- Creado por Guido van Rossum en 1990 durante sus vacaciones
- Fan de los Monty Python
- Multiparadigma: funcional, orientado a objetos, minimalista, imperativo, scripting, etc.
- Interactivo de prototipado rápido.
- Open Source Project (Python License, compatible GPL).
- Managed by non-profit Python Software Foundation (PSF).



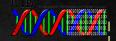


http://www.python.org

"Python is a dynamic object-oriented programming language that can be used

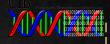
for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days. Many Python programmers report

substantial productivity gains and feel the language encourages the development of higher quality, more maintainable code."



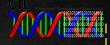
- Lenguaje scripting pero también proyectos grandes.
- Gran manejo de cadenas, listas, tuplas, y diccionarios
- Fully dynamic type
- Libera de la preocupación por la memoria (automatic memory management).
- Gran cantidad de operaciones nativas.
- Módulos (numerosos, fácil uso, bien documentados)
- Espacios de nombres, clases, excepciones, etc.

Python



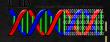
- Extensible (C, C++, Java).
- Empotrable en aplicaciones.
- Portable (Unix, Windows, Mac, AS/400, PalmOS, PlayStation, etc...).
- > Interpretado (bytecode).
- Gestión de memoria (referencias + Garbage Collection).

450



- Prototipado rápido
- Scripts, aplicaciones web, control de aplicaciones, procesamiento
 XML, bases de datos, interfaces gráficos
- Muy usado en educación y ciencia
- ¿Quién usa Python?
 - Google, Yahoo
 - Zope, plone, mailman, etc.
 - NASA, NYSE (bolsa de N.Y.), ILM (Industrial Light & Magic)

Instalación

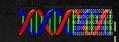


- > Python v3.7.4 (16 julio 2019)
- http://www.python.org/download/
- Fuente o binarios
- Linux
- > Generalmente está ya instalado, o la distribución incluye paquetes:
 - apt-get install python python-doc
- Unices (UNIX)
- Hay binarios disponibles
- Se puede compilar

Walling Service Control Contro

Ejecución

- Modo interactivo, ejecutar en la shell la orden:
 - python
- Creando un script
- Modo 1 de ejecutar el script:
 - python [opciones] knombre.py>
- Modo 2 de ejecutar el script:
 - #! /usv/bin/env python
 - # -*- coding: utf8 -*-
 - chmod +x nombre.py
 - ./nombve.py

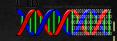


Modo interactivo

```
Algo interesante/útil:
>>>import rlcompleter, readline
|>>>readline.parse_and_bind("tab:complete")
probar entonces:
>>>import math
>>>math.(pulsar <TAB> dos veces)
math.acos math.e
                       math.pi
math.asin math.fabs math.pow
math.atan math.log math.sin
math.cos math.log10 math.tan
```

>>>math.

O más cómodo con un script de inicio siguiendo los pasos: Crear el fichero "pythonrc.py" con las dos líneas anteriores La variable PYTHONSTARTUP debe tener el camino al script: export PYTHONSTARTUP="/..../ .../pythonrc.py"



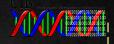
Modo interactivo

```
Ayuda interactiva:
>>>help # breve mensaje de ayuda general
>>>help(objeto) # ayuda de ese objeto
>>>help("if") # ayuda de if
>>>help() # Ayuda interactiva
help>
help>keywords
help>modules
help>topics
help>LISTS
help>print()
help> CTRL+D (o quit)
>>>
```

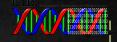
el sistema de ayuda es muy importante en Python y los desarrolladores deben impregnarse de él ... y continuarlo en sus programas.

09/09/20 11/93

IDLE



- Editor oficial del proyecto:
 - IDLE is the Python IDE built with the Tkinter GUI toolkit.
- > Python shell (abre un shell).
- Check Module (comprueba la sintaxis sin ejecutar).
- Run Module (ejecuta el programa, lo salva previamente).
- Restart shell: CTRL+F6
- si hay algún objeto global o variable definida los elimina.
- o un módulo importado, etc.
- Más en http://www.python.org/idle
- > Se puede usar cualquier otro editor de textos



Primeras instrucciones

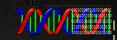
```
No hay terminadores de línea
";" separa instrucciones en una misma línea
'print("hola")
print("hola"); print("y adiós")
print ("hola"), #no salta línea
Unión de líneas:
Explícita: con \ como en:
   a= 2*pi \
Implícita: dentro de (, { o l, como al definir la siguiente lista:
   lista= [12, 43, -12]
      , 32, 1,
```

09/09/20

1029]

Comentarios

- Con el símbolo "#" hasta fin de línea
- cometarios y autodocumentación son muy importantes....



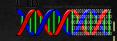
Calculadora

- El intérprete evalúa literales, variables, expresiones interactivamente.
- Operadores: +,-,*,/,**,%,()
- Avance del uso de funciones matemáticas (el módulo math):
- >>>import math
- >>>help(math)
- >>>help(math.pow)
- Otras: math.log(), math.log10(), math.sin(), math.cos(), math.pow(), math.tan(), math.fabs(), math.pi, math.e, etc...
- El guión bajo "_" es el último valor evaluado en el intérprete.



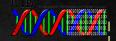
Modelo de datos de Python

- Es más complicado explicarlo que usarlo...
- En Python todo son objetos con:
 - id único entevo (no cambia, lo devuelve la función id(x))
 - tipo (no cambia, lo devuelve la función type(x))
- Puedo asignar un nombre temporalmente a un objeto:
 - a=1 la vaviable "a" es una vefevencia al objeto entevo (int): 1
- A continuación puedo hacer: a=3.5
 - la vefevencia apunta ahova al objeto veal (float): 3.5
- > Cada objeto tiene sus operaciones y algunos no pueden cambiarse:
 - objetos inmutables: números, cadenas y tuplas
 - objetos mutables: listas y diccionavios
 - El objeto entevo I no puede cambiavse, dejavía de sev el I
 - Una lista de elementos puede modificarse y en un instante posterior la misma lista tener otros elementos diferentes



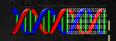
Modelo de datos de Python

- Se evalúa la expresión a la derecha del igual resultando un objeto. Python reserva una celda de memoria para dicho objeto.
- Se hace que la parte izquierda del igual apunte al resultado (haga referencia a ese objeto).
- La variable a la izquierda del igual no tiene tipo fijo, puede hacer referencia a un entero y luego a un real.
- El valor de la derecha si tiene tipo
- Python es dynamic typed/loosely typed



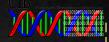
Modelo de datos de Python

- Un objeto puede tener cero o más nombres.
- Un nombre es una entrada en el espacio de nombres y hace referencia a un objeto. Puede hacer referencia a otro cuando quiera.
- Una asignación significa que una entrada del espacio de nombres es asociada a un objeto (hace referencia a un objeto).
- Si el objeto es mutable, podré acceder a métodos que lo hagan cambiar (listas, diccionarios). También cuando se reciben como parámetro de una función.
- Si el objeto es inmutable, no habrá métodos que lo cambien (numeros, cadenas, tuplas). Tampoco cuando se reciben como parámetro de una función.



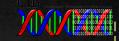
Identificadores

```
Python es case-sensitive
   (letral"_") (letra | dígito | "_")*
  *: no se importan de los módulos (from module import *)
     *__: identificadores del sistema. Ej:
__name__
__doc__
__init__()
__del__()
__str__()
__*: nombres privados de clase
```



Números

- > Enteros:
 - Rango: -2147483648 ... 2147483647 (32 bits, sys.maxint)
 - Octal: 0177
 - Hexadecimal: 0xFF, 0XFF
- Enteros largos: rango ilimitado, precisión ilimitada
 - terminan con L o I o bien usando long()
- Lógicos o booleanos: True (1) y False (0) (case sensitive)
 - Cualquier entero (positivo o negativo) distinto de cero es True.
 - 0 es False (también lo es la lista, tupla, cadena, etc. que estén vacías).



Números

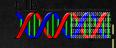
- Float (son de doble precisión): 1.14e-10,.001,1.,1E3 (siguen la codificación del estándar IEEE 754)
- Complejos: 1J, 2+3J,4+5j
- > Operadores: +, -, *, /, +=, -=, *=, /=, **, %, >, <, >=, <=, ==, !=, and, or, not, (), <<, >>, |, &, ^
- No hay ++, --
- Precedencia habitual: PEDMAS (Parentheses, Exponentiation, Multiplication/Division, Addition/Subtraction)
- Recordar:
 - La función type(...)
 - Los objetos números son inmutables

Números



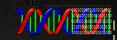
```
abs()
coerce(5,2L) -> (5L,2L) (Deprecated)
divmod(5,2) \longrightarrow (2,1)
pow(5,2) -> 25
round(x[,n])
round(5.567) -> 6
round(5.567,2) -> 5.57
\min(x,[y,z...])
\max(x,[y,z...])
cmp(x,y)
-l si x < y
O si x=y
l si x>y
```





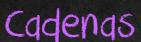
> Asignaciones

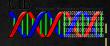
Incrementos



Cadenas

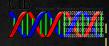
- En Python el manejo de cadenas (strings) se dice que es nativo al lenguaje (cadenas nativas).
- Es decir, forman parte del lenguaje (no se hacen vía otra librería u otro tipo como en C, C++, etc.).
- Python es muy muy potente en el manejo de cadenas





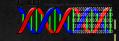
```
Ejemplos:
!>>>fruta="platano" #o bien fruta='platano'
|>>>letra=fruta[1]
>>>print(letra)
>>>len(fruta)
>>>print(fruta[len(fruta)-l])
>>>print(fruta[-3])
a (tercero empezando por el final)
>>>print(fruta[-35])
Traceback (most recent call last):
File "<stdin>", line l, in ?
IndexError: string index out of range
```

Cadenas



```
string slices (slicing):
>>>s="Juan, Pedro y María"
>>> print(s[0:4])
Juan
>>>print(s[6:l1])
Pedro
>>>print(s[:11])
Juan, Pedro
>>>print(s[12:])
y María
```

No hay tipo char, es una cadena de l elemento



Cadenas

```
componer cadenas literales:
el lenguaje "Python" fue escrito por Guido van Rossum'
"el lenguaje "'Python' fue escrito por Guido van Rossum"
>>>s= """una
   cadena larga"""
>>>s= "me llamo_\
>>>... juan"
'me llamo juan'
Escapes como en C: \n, \t...
Cadenas 'crudas': (no interpreta los escapes)
   >>> print("\t hola")
         hola
    >>> print(r"\t hola")
    \t hola
```





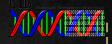
Las cadenas son secuencias inmutables:

```
|>>>s="juan"
|>>>s[O]=']' #ERROR
|>>>t=']'+s[l:]
|>>>print(t)
|Juan
```

Razones:

- Cuestiones de eficiencia, gestión sencilla de memovia.
- Se consideran tan fundamentales como los números.
- No se pueden cambiar, sí veasignar.





- Comparación: =, >, <,>=,<=,!=</p>
- Operaciones +, *:

```
'>>>s="hola"
```

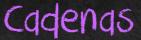
$$>>>print(s+t)$$

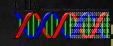
hola y adios

$$>>>print(s*3)$$

holaholahola

Concatenación +, +=, *=





```
hola upper() -> "HOLA" (lower)
hola strip() -> "hola"
```

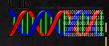
Formateo de cadenas:

"hola %s, son las %d" % ("juan", 5)
s,d,f,c,u, etc., + delante siempre pone el signo
x.y (y decimales de los x totales)

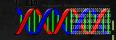
OPERACIONES SOBRE CADENAS

```
bool("hola") -> True
bool("") -> False
max("abcde") ->e
min("abcde") ->a
max("juan", "antonio", "pedro") ->pedro
min("juan", "antonio", "pedro") ->antonio
"a" in "juan" -> True
a not in "juan" -> False
```



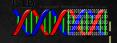


```
>> int("lO")
>>>int (3.7)
>>>int("lO.3")error, sería: int(float("lO.3"))
round(num[,digits])
long()
float()
ord(num) -> ASCII o Unicode de num
La inversa: chr() unichr()
oct(), hex()
str()-> convierte cualquier cosa a una cadena
```



Módulo "string"

- >>>import string
- >>>string.letters
- "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUV WXYZ"
- >>>string.lowercase
- "abcdefghijklmnopqrstuvwxyz"
- >>>string.uppercase
- "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
- >>>string.digits (octdigits, hexdigits)
- "Ol23456789"
- >>>string.punctuation
- "!"#\$%&\\\()*+,-./:;<=>?@[\\]^_\{|}~"



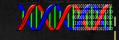
Módulo "string"

```
import string
string.split(cadena) # crea una lista con las palabras
                             # elimina espacio, newline, tab, etc.
                    # se puede usar cualquier separador
también se puede:
cadena.split()
Otras:
string.find(cadena,s,[,start][,end]) -> devuelve el índice en el que
empieza "s
string.replace(cadena,old,new)
string.join(lista,sep) -> devuelve una cadena formada por las
palabras de la lista separadas por el separador dado
```

None

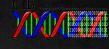


- Hace referencia a Nada (NULL en C/C++).
- Las funciones que no devuelven nada devuelven None
- > Su valor de verdad es False



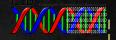
Funciones

- Built-in: type(), id(),int(),float(),str()
- Matemáticas:
- >>>import math
- >>>help(math)
- >>>help(math.pow)
- Otras: math.log(),math.log10(),math.sin(),math.cos(), math.pow(),math.tan(),fabs(), etc...
- Operador punto: acceso a atributos y métodos de los objetos nombre.atributo
 nombre.metodo()



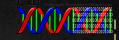
```
if x>0:
   print("x es positivo")
if x\%2 == 0:
   print(x, " es par")
else:
   print(x, " es impar")
if x < y:
   print(x, "es menor que", y)
elif x > y:
   print(x, "es mayor que", y)
else:
   print(x, "y", y, "son iguales")
```

No se requieren paréntesis en la expresión booleana No hay switch



I/O estándar

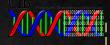
```
Salida estándar
   print('Hola,', 'Mundo')
   print('Uno ', 'Dos')
   print('%s-%d.txt' % (nombre, num))
   print('uno', ; print('dos' (no salta línea)))
Entrada estándar
   ent= int(raw_input('Dame un número: '))
   cad= raw_input('Dame una cadena: ')
   real = float(raw_input())
```



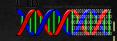
Conversión de Lipos

- bool(0); bool(123); bool(-12)
- int('123')
- float('3.1415')

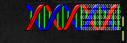
09/09/20 38/93



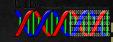
```
Definidas por el usuario antes de su uso
def NOMBRE(LISTA DE PARÁMETROS):
  INSTRUCCIONES
Ejemplo:
def fun(a, b):
   return a+b #su definición no ejecuta la función
x = fun(3, 2)
Cadenas de documentación (docstring):
def fun(a, b):
   "Suma dos números"
                                             Docstrina
   return a+b
print(fun.__doc__) # (print docstring)
```



- Terminamos una función con return
- Funciones python:
 - No hay tipo de la función o tipo de valor devuelto.
 - No hay tipo de los pavámetros, solo lista enumerándolos.
 - Si no devuelve nada, devuelve None.
- Puede devolver varios parámetros (ver tuplas, listas, etc.)

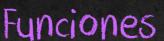


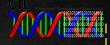
- No hay ficheros de declaración (.h)
- Hay recursión
- Argumentos con valores por defecto igual que en C++
- Los parámetros se pasan by assignment:
 - Los pavámetvos formales vefevencian a los objetos que llegan:
 - Si son inmutables NO quedan modificados.
 - Si son mutables SI quedan modificados.



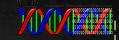
Variables globales

- Las declavadas en una función son locales
- Las declavadas fueva son globales
- Dentro de una función uso: global var
- o pava indicav que "vav" es global.





```
Número variable de argumentos
def fun(i, *args):
   print(" i=",i)
   for arg in args:
      print(arg, "tipo = ", type(arg))
>>>fun(1,2,3,4,5.6,"hola")
i=1
2 tipo = <type 'int'>
3 tipo = <type 'int'>
4 tipo = <type 'int'>
5.6 tipo = <type 'float'>
hola tipo = <type 'str'>
```



Introducción a los modules

- Para incluir (parecido a #include) otro fichero Python con declaraciones se usa:
 - import fichero
- Se carga el fichero.py y el código de ese fichero que no forma parte de una función o una clase se ejecuta.
- Para adosar código que se ejecute sólo si el módulo se invoca directamente:

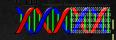
```
if (__name__==__main__):
```

•••

...

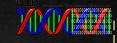
Sucesivos "import" no tienen efecto (para no volver a cargar lo mismo en distintos sitios de un programa).

Para forzar la re-carga: reload(modulo)



Test de Lipos

```
def factorial(n):
   if type(n) = type(1):
      print ("factorial está definida solo para enteros")
      return -l
   elif n < 0:
     print ("factorial está definida solo para enteros positivos")
     return -l
   elif n = 0
      return l
   else:
      return n * factorial(n-l)
```



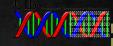
command line arguments

```
import sys
print(sys.argv)
sys.argv[0] -> string con el nombre del programa
sys.argv[1] -> string con ler par. línea de comandos
...
```

sys.argv[n] -> string con n-ésimo parámetro

Método más avanzado: import argparse



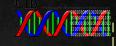


```
def cuentaAtras(n):
    while n > 0:
        print (n)
        n=n-l
        print ("fin")

def multiplos(n):
    i=l
    while i <= 10:
        print (n*i,'\t')
        i=i+l
    print ("fin")</pre>
```

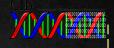
No se requieren paréntesis en la expresión booleana

Listas



```
Secuencias mutables
[1] #es una lista
[1,2,3] # es una lista
[] #es la lista vacia
   Asignando (de cualquier tipo e incluso híbridas):
ll = []
12=[1]
13 = [0,5,10,15]
14=[1,4,1,'esto', False, -5]
> Slices:
[13[0:2] # resultado: [0,5]
13[:] # resultado: [0,5,10,15]
[13] # resultado: [5,10,15]
13[:3] # resultado: [0,5,10]
```

Listas



Modificando elementos:

```
|3[0]=77 #mutable. Fuera de índice da ERROR | 13[0]=[1,2] -> [[1, 2], 5, 10, 15] #ojo con esto
```

Modificando slices (sustituye primer slice por el segundo):

```
13[O:2]=[1,2]
13[O:O]=[1,2,3,4]
13[O:4]=[1]
```

13[0:25]=[1] #fuera de rango, funciona y cambia la lista por [1]

Añadiendo una lista al final:

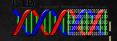
l3.extend(x) # Extiende la lista con otra lista l3.append(x) # Añade un elemento al final l3.insert(i,x) # Inserta x antes del elemento i-ésimo



```
Note the second second
```

```
Información:
len(13)
13.count(x) # cuenta el número de x en la lista
13.index(x,[start[,stop]])) #menor i tal que
                       # 13[i] = x entre start y stop
   Borrar:
del 13[1] #borra el segundo
del 13[-1] #borra el último
[3.remove(x) \# iqual del [3][13.index(x)]
Operaciones:
13.reverse() # in-place (no devuelve una lista, ordena 13)
```

l3.sort() # in-place (no devuelve una lista, ordena l3)

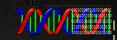


Listas como pilas

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
 >>> stack
[3, 4, 5, 6, 7]
 >>> stack.pop()
 >>> stack
[3, 4, 5, 6]
>>> stack.pop()
 >>> stack.pop()
 >>> stack
\begin{bmatrix} 3, 4 \end{bmatrix}
```

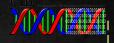
Con parámetro: l.pop(i)

x=l[i];del l[i];return x



Listas como colas

```
>>> queue = ["Eric", "]ohn", "Michael"]
>>> queue.append("Terry") # Terry arrives
>>> queue.append("Graham") # Graham arrives
>>> queue.pop(0)
'Eric'
>>> queue.pop(0)
'John'
>>> queue
['Michael', 'Terry', 'Graham']
```



copiar una lista

```
a=[1,2,3]
b=a # el nombre "b" hace referencia al mismo objeto
# al que hace referencia el nombre "a"
# jijCuidado!!!
b=a[:] # copia, a y b son objetos diferentes
```

a =[una lista muy complicada de varios niveles...]
import copy
b=copy.deepcopy(a)# Copy all levels, avoid side effects

Listas



- Funciones que devuelven listas
 - Generando listas (para bucles 'for'):

```
range(10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
range(1,5) [1, 2, 3, 4]
range(0,10,2) [0, 2, 4, 6, 8]
```

Listas a partir de cadenas:

```
>>> "juan ana pedro".split()
["juan","ana","pedro"]
```

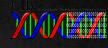
>>> "natalia:alfredo:julia".split(":")

```
["natalia", "alfredo", "julia"]
```

List:

$$l = list()$$





```
Vectores:
```

$$v = [2,4,6,8,10]$$

Matrices:

```
    2
    4
    5
    6
    8
    9
```

```
>>>matriz=[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

>>>matriz[l]

>>>matriz[1][1]

5





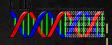
56/93

```
for i in range(10):
     print(i)
for i in range(l,ll):
     print(i)
cadena="platano"
for i in cadena:
     print(i)
lista=["juan","maria","pedro","eva"]
for i in lista:
```

Hay break, continue, pass (no hace nada, placeholder)

print(i)

For



Sobre secuencias: cadenas, tuplas, listas, diccionarios (claves), ficheros (líneas):

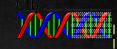
```
for c in persona:
print (c, ':', persona[c])
```

NO COMPOSITION OF THE PROPERTY OF THE PROPERTY

Tupla

- TUPLE: SECUENCIA DE VALORES SEPARADOS POR ","
- itupla='a','b','c' tl=('a',) #sin la ",", tl es un str
 - LAS TUPLAS SON INMUTABLES.
 - SLICES IGUAL QUE EN LISTAS.
 - ASIGNACIÓN:
- >>> a,b,c = 1,2,3
- >>>size = width, height = 320, 240
- >>>size
- (320,240)
- >>> width
- 320
- >>>height

Tuplas

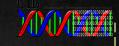


> Otros usos:

> mejor:

$$a,b = b,a$$

Tuplas

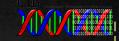


Tuplas como valores de retorno de funciones: def swap(x,y): return y, x

$$a, b = swap(a, b)$$

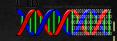
Se pueden asignar valores mezclando listas y tuplas:

```
(a, b, c) = "foo:bar:baz".split(':')
```



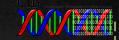
Tuplas vs. listas

- Varios valores de un mismo objeto: tuplas
 - Ej: coovdenadas de un punto (x,y,z)
- Procesamiento de muchos elementos del mismo tipo: listas
- Líneas de un fichero de entrada: listas
- Elementos distintos: tuplas
- Diferentes partes de un mismo elemento: tuplas



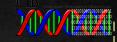
conversion lista + tupla

- La función list(): crea una lista de una tupla
- La función tuple(): crea una tupla de una lista



Secyencias

- Para todas las secuencias: cadenas, tuplas y listas
 - indexadas con []
 - soportan slicing
 - valov [not] in secuencia
 - secuencia + secuencia
 - secuencia * vepeticiones
 - len(secuencia)
 - min(secuencia)
 - max(secuencia)
 - sum(secuencia) # si lo admiten sus elementos



conjuntos

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket) # create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])
>>> 'orange' in fruit # fast membership testing
True
>>> 'crabgrass' in fruit
False
>>> # Demonstrate set operations on unique letters from two words
>>> a = set('abracadabra')
>>> b = set('alacazam')
                                 # unique letters in a
>>> a
set(['a', 'r', 'b', 'c', 'd'])
                                 # letters in a but not in b
>>> a - b
set(['r', 'd', 'b'])
                                 # letters in either a or b
>>> a | b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
                                   # letters in both a and b
>>> a & b
set(['a', 'c'])
>>> a ^ b
                                  # letters in a or b but not both
set(['r', 'd', 'b', 'm', 'z', 'l'])
```



Diccionarios (mapas)

Valores (values) indexados por índices (keys) arbitrarios.

```
En este grupo únicamente está el diccionario (dict):
'>>>persona= {'nombre':'Pedro', 'edad':25, 'casado':True}
>>>persona
{'edad': 25, 'nombre': 'Pedro', 'casado': True}
>>>persona['nombre']
Pedro
```

```
>>>persona['notas']= [None, None, None]
```

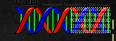
>>>persona

{'edad': 25, 'nombre': 'Pedro', 'casado': True, 'notas': [None, None, None]}

>>>persona['notas'][0]= 5.8

>>>persona

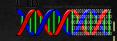
{'edad': 25, 'nombre': 'Pedro', 'casado': True, 'notas': [5.8, None, None]}



Diccionarios

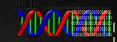
- Los diccionarios son mutables
- Operaciones:

```
len(persona)
del persona['edad']
persona.has_key('peso')
persona.items() # lista de tuplas (key,value)
persona.keys() # lista de keys
persona.values() # lista de values
```



Excepciones

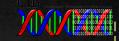
- Una excepción es un objeto Python que representa un error en tiempo de ejecución
 - Python: esquema try...except, raise
- Built-in exceptions: ZeroDivisionError, NameError, TypeError
- User-defined exceptions



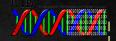
Excepciones

- Cuando en una función se eleva una excepción:
 - debe manejavla o termina
 - o bien debe manejavla su callev o terminav (así sucesiv.)
 - o bien debe terminar el programa

```
try:
   bloque
[except [e...]]:
   bloque
[else:
   bloque]
[finally:
   bloque]
```



- A finally clause is always executed before leaving the try statement, whether an exception has occurred or not.
- When an exception has occurred in the try clause and has not been handled by an except clause (or it has occurred in a except or else clause), it is re-raised after the finally clause has been executed.
- The finally clause is also executed "on the way out" when any other clause of the try statement is left via a break, continue or return statement.

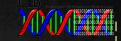


Excepciones

Ejemplo:

```
>>> def divide(x, y):
        try:
            result - x / v
        except ZeroDivisionError:
            print("division by zero!")
        else:
            print("result is", result)
        finally:
            print("executing finally clause")
>>> divide(2, 1)
result is 2.0
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

09/09/20 70/93



Excepciones

- Python muestra en cada error.
 - La función o funciones que la elevan (most vecent calls last)
 - El contexto (fichevo y línea del código)
 - La excepción elevada y algún detalle/comentavio
- Ejemplos:

Traceback (most recent call last):

File "<stdin>", line l, in?

ZeroDivisionError: integer division or modulo by zero

$$>>> 4 + spam*3$$

Traceback (most recent call last):

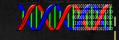
File "<stdin>", line l, in?

NameError: name 'spam' is not defined

Traceback (most recent call last):

File "<stdin>", line l, in?

TypeError: cannot concatenate 'str' and 'int' objects



Clases

```
class Prueba:
   """Esto es una clase de prueba"""
  nObjetos=0 #variable de clase
  def __init__(self,nombre):
     """constructor de la clase"""
     self.n=nombre + #variable de instancia
     Prueba.nObjetos += 1 #uso de variable de clase
      print("creada instancia %d de la clase Prueba" % Prueba.nObjetos)
  def muestraNombre(self):
     """muestra el nombre del objeto y el de la clase"""
     print("nombre objeto = ", self.n)
     print("nombre clase = ", self.__class__)
>>>o=Prueba("juan")
creada instancia 1 de la clase Prueba
>>>p=Prueba("ana")
creada instancia 2 de la clase Prueba
o.muestraNombre()
o.__dict__
o.__class__
                                     nombre de la instancia
                                    dentro de la clase (=this)
```

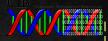
atributo de clase *Prueba.* doc

constructor

"self" es obligatorio en todos los métodos

instanciación

Clases



```
Datos privados:
```

variable estática en C++

def Prueba:

datoPublicoclass=4 #variable de clase pública

miembro público en C++

def fl(self):

self n=1 #variable de instancia pública

siempre self

miembro privado en C++

self.__datoPrivado=7 #variable de instancia privada

return Prueba.datoPublico, self.__datoPrivado

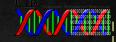
```
>>>o=Prueba()
```

(4,7)

>>>o.n -> acceso OK

>>>o.__datoPrivado -> acceso ERROR

siempre nombre de la clase



Clases

```
>>>Prueba.nvar=55 # creando una nueva variable de clase

>>>o.nvar

55

>>>p.nvar

55

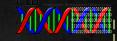
>>>o.nnvar=5 #creando una nueva variable de instancia

>>>o.nnvar # existe

55

>>>p.nnvar # no existe
```

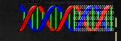
ERROR...



Clases

Variables:

- De clase (nombreClase.nombreVariableDeClase). Para todas las instancias de la clase.
- De instancia (self.nombre Variable Instancia). Para todos los métodos de la instancia.
- Locales a los métodos (nombre VariableLocal). Únicamente en el método que la declare.



Herencia

```
class Vehiculo:

velocidadMaxima=120

def acelera(self, a):

print("más rápido", a)

def frena(self):

print("para!!")
```

```
class Camion(Vehiculo):

velocidadMaxima=100

def carga(self, c):

print("mi carga es ", c)

def frena(self):

Vehiculo.frena(self)

print("frenazo de camión!!")
```

```
Ejecución:

>>>c=Camion()

>>>c.acelera(60)

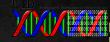
más rápido 60

>>>c.frena()

para!!
```

frenazo de camión!!

Herencia



- Hay herencia múltiple
- No hay sobrecarga de funciones en Python
 - Dos funciones en el mismo ámbito no pueden llamavse igual aunque tengan distintos pavámetvos, etc.
 - Sí en distintas clases, módulos, etc.



Sobrecarga de operadores

```
class Contador:
           """La clase contador es un ejemplode sobrecarga de operadores
           HIIII
           def __init__(self,val):
              self.n=val
           def __add__(self, obj):
              print(self.n + obj)
           def __radd__(self, obj):
              print(self.n + obj)
           def __str__(self):
              return "contador = %d" % self.n
add(x+y)
iadd (x+=y)
radd (x+y) x no tiene operador add (igual que invocando: y_{-}radd_{-}(x))
Igual en sub, mul, div ...
No se puede sobrecargar la asignación (en Python asignar es enlazar
un nombre a un objeto)
```

Page 100 to 100

Módulos

- Los módulos son ficheros .py con definiciones (variables, funciones, clases, etc.) e instrucciones python.
- Se pueden importar a cualquier otro módulo o al módulo principal.
 import modulo [as alias]
 - Supongamos el módulo funciones.py: def fun1(n) y def fun2(n):
- Dentro de otro fichero o en el intérprete directamente:

```
import funciones
```

funciones.funl(5)

funciones.fun2(7)

....

fl=funciones.funl

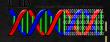
f2=funciones.fun2

....

fl(5)

f2(7)

Módulos



- import suele ponerse al principio del fichero, aunque se puede poner en cualquier sitio.
- Los módulos se buscan en el directorio actual y en los indicados en la variable de entorno: PYTHONPATH
- Además en el installation-dependent default path, que en debian, por ejemplo, es /usr/lib/python
- > Se puede acceder a esta lista desde el programa:

import sys

sys.path

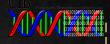
Si hay módulos en otros directorios, se debe crear un fichero con la extensión .pth en /usr/lib/python2.4/site-packages indicando en cada línea, el directorio donde hay módulos.

to the control of the

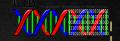
Módulos

- > import además de cargar el módulo lo ejecuta
- Pero sucesivos import no tienen efecto por si en distintos lugares del código se carga el mismo módulo (sería costoso en tiempo y memoria)
- Para forzar la recarga (y la ejecución): reload(modulo)

Módulos

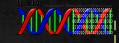


- Importando partes de un módulo from modulo import (*|lista separada por comas)
 - Así se puede invocar directamente las funciones (o variables, clases, etc.) importadas sin poner el nombre del módulo delante.
 - No es elegante abusar de: from modulo import *
 - No abusar para no machacar nombres del espacio de nombres actual.
 - Se usa con frecuencia en el intérprete para ahorrarnos 'tecleo'.



Módulos (bytecode)

- Si existe el fichero nombre.pyc, es la versión 'byte-compiled' del programa nombre.py
- Cuando nombre.py se compila al hacer un import se intenta crear el fichero nombre.pyc
- Este bytecode es independiente de la plataforma.
- Los módulos compilados pueden compartirse entre plataformas.
- No son más rápidos, simplemente se cargan antes.
- Cuando un script se ejecuta directamente en la shell, no se crea el .pyc, por ello es mejor crear un pequeño 'programita' en python que lo importe (un bootstrap).



Módulo "random"

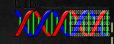
import random

random.random() #devuelve un float en el intervalo [0,1)

random.uniform(a,b) #devuelve un float en el intervalo [a,b)

random.choice(lista) # escoge un elemento al azar

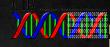
random.choice(string.letters)



85/93

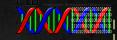
- Apertura con open():
- =open("nombre", "modo")
- modo:
 - v lectuva
 - w escritura (destruye si ya existe)
 - a añadir (crea uno si no existe)
 - v+ lectura y escritura (debe existir)
 - w+ lectuva y escritura (destruye si existe)
 - a+ lectura y añadir (crea uno si no existe)
- Si añadimos:
 - t fichevo de texto
 - b fichevo binavio





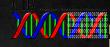
Atributos:

```
>>>f=open("prueba.txt", "wt")
>>>f.mode
'wt'
>>>f.closed
O
>>>f.name
'prueba.txt'
>>>f.write("hola en el fichero")
>>>f.close()
```



Posición:

```
>>>f=open("prueba.txt", "w+")
>>>f.write("CARLOS")
>>>f.tell()
6 # la próxima escritura será en la posición 6
>>> f.seek(2) # se mueve al offset 2
>>>f.write("rl")
>>>f.seek(0) # va al comienzo
>>>f.read() #lee todo, desde actual hasta el final
CArlOS
>>>f.tell()
6
```



Posición:

```
fseek(position, whence)

position= offset

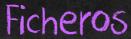
whence=0, desde el principio (por defecto)

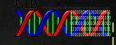
whence=1, desde actual

whwnce=2, desde el final
```



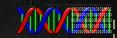
```
Líneas:
     >>>lineas=["primera linea","segunda","final"]
     >>>f.writelines(lineas)
     >>>f.seek(0)
     >>>f.read()
     primera linea
     segunda
     final
     >>> for i in range (\overline{3}):
           f.write("fila %d \n" % i)
     >>>f.seek(0)
     >>> f.read(3)
     'fil'
     >>>print(f.read())
     >>>f.seek(0)
      >>>f.readline()
      'primera fila'
      >>>f.seek(0)
      >>>f.readlines()
     ["primera linea", "segunda", "final"]
```





Lectura rápida del fichero:

```
f=open("fich.txt","r")
for i in f:
    print(i)
```

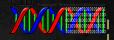


List comprehension

lista=[i for i in range(1,11)]

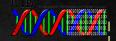
def multiplicar(x,y): return x*y

l=[(x,y) for x in (l,4,6,24,l9) for y in (l5,7,l,2) if multiplicar(x,y)>25]



Expresiones regulares

```
Potentes y mejor forma de manipular cadenas
import re
'cad= "dabale arroz a la zorra el abad"
expc = re.compile("(l.*?a)")
matchobjl= expc.search(cad)
if matchobil:
  print(matchobjl.groups())
matchobj2= expc.match(cad)
if matchobj2:
  print(matchobj2.groups())
print(re.compile('rr').sub('--',cad))
```



Expresiones regulares

Los caracteres y combinaciones especiales en ER:

```
^ $ * + ?

*? +?

{n} {n,m}

[...] [^...]

|

(...)
```

Más en

http://www.amk.ca/python/howto/regex/

Tutoriales

https://docs.python.org/3/tutorial/

https://www.w3schools.com/python/default.asp