

Introducción a la programación

Python

resumen

Profesorado Don Bosco

Índice

- 1 - Entrada / salida por consola - condicionales - lazos -
- 2 - Secuencias (Strings, Listas, Tuplas) y Diccionarios
- 3 - Funciones y módulos
- 4 - Excepciones
- 5 - Archivos de texto
- 6 - JSON
- 7 - Funciones anónimas / map / filter / comprensión de listas
- 8 - SQLite
- 9 - GUI
- 10 - Framework WEB (Bottle)

Algunas características

- Interpretado (JIT Compiler)
- "Write once, run many"
- Tipado dinámico
- Multiparadigma (Estructurada, Objetos)
- Muchas bibliotecas y recursos
- Shell Interactivo (REPL)
- Relativamente lento

El lenguaje Python - estructura

```
#!/usr/bin/python3      ← qué usar para ejecutar el script
# -*- coding: utf-8 -*- ← codificación del archivo

def main():
    print("Hola!")

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

...o simplemente:

```
print("Hola!")
```

El lenguaje Python - estructura

```
# determina el mayor de dos números
opc = 's'
while opc == 's':
    a = int(input("Ingrese el primer nro.:"))
    b = int(input('Ingrese el segundo nro.:'))
    if a > b:
        print("El primero es mayor que el segundo")
    else:
        print("El segundo es mayor que el primero")
    opc = input("otro? (s / n)")
```

- ¡Las sentencias se agrupan según el nivel de indentado!
(En otros lenguajes se usa {} o begin/end o...)

El lenguaje Python - identificadores

- **Nombres válidos de identificadores:**
 - Sucesión de letras, dígitos y _
 - No puede comenzar en dígito.
 - Se distingue entre mayúsculas y minúsculas
 - De cualquier longitud.
 - No puede coincidir con el nombre de una palabra reservada:

`and, del, fo, is, raise, asser, elif, from, lambda, return, break, else,
global, not, try, class, except, if, or while, continue, exec, import, pass,
yield, def, finally, in, print`

Ej.: `importeTotal, resultado, _, cant_veces, cod12, Complejo`

El lenguaje Python - variables

- Las variables en Python no se declaran (adoptan el tipo correcto al asignarles un valor)
- Antes de mostrarlas o usarlas en una expresión tienen que estar inicializadas: deben tener un valor definido.

```
print(a) # NameError: name 'e' is not defined
```

```
a = 5  
print(a * a) ← muestra 25
```

El lenguaje Python - variables

- algunos "tipos": bool, str, int, float
- ...pueden "cambiar" de tipo... (pero es confuso, evitarlo)

```
a = "una frase"
```

```
print(a)
```

← muestra una frase

```
a = 5
```

```
print(a * a)
```

← muestra 25

Salida por consola

- Se utiliza `print()`

<code>print("hola")</code>	← muestra hola
<code>print(a)</code>	← muestra el valor de a
<code>print(2 * 4)</code>	← muestra 8

- Se puede especificar qué se agrega al final (en lugar de salto de línea) y/o separador...

<code>print("uno dos", end="***")</code>	uno dos***
<code>print("uno", "dos", sep="####")</code>	uno####dos

Entrada por consola

- Se utiliza `input()`

```
a = input()  
a = input("ingrese nombre")
```

- `input()` devuelve un string... (si por ejemplo se ingresa 132, lo devuelve como string "132")
- Para convertir un string a número:

```
n = int(input())    → como entero  
n = double(input()) → como número de punto flotante
```

Expresiones lógicas

Condiciones:

`a == b` igual

`a != b` distinto

`a < b` menor

`a <= b` menor o igual

`a > b` mayor

`a >= b` mayor o igual

`not(expr)` not niega la expresión $\rightarrow$ `not(True)` es `False`

Condiciones compuestas:

`(b > 3) and (b < 8)` and: True si todas las condiciones True

`(b >= 4) or (b <= 2)` or: True si al menos una condición True

Condicionales

- Las estructuras condicionales permiten ejecutar diferentes grupos de instrucciones de acuerdo al resultado de una expresión lógica → "condición"
- `if` es la estructura condicional disponible en Python:

```
if [condición] :  
    sentencias
```

```
if [condición] :  
    sentencias  
else:  
    sentencias
```

Condicionales

```
if tipo == 0:
    print("efectivo")
else:
    if tipo == 1:
        print("débito")
    else:
        if tipo == 2:
            print("crédito")
        else:
            print("otro")
```

- Para evitar "anidar" muchos if se puede usar `elif`

```
if tipo == 0:
    print("efectivo")
elif tipo == 1:
    print("débito")
elif tipo == 2:
    print("crédito")
else:
    print("otro")
```

Condicionales - ejemplos

```
num = int(input())  
if num % 2 == 0:  
    print("par")  
else:  
    print("impar")
```

```
recargo = double(input())  
if recargo >= 0:  
    precio *= recargo
```

```
lados = int(input())  
if lados == 3:  
    print("triángulo")  
elif lados == 4:  
    print("cuadrilátero")  
elif lados == 5:  
    print("pentágono")  
else:  
    print("muchos lados")
```

Lazos (Loops)

Permiten repetir grupos de sentencias. En Python:

- **while:** Mientras una condición sea verdadera

```
while [condición]:  
    sentencias
```

- **for:** Para cada uno de los elementos de un iterable

```
for variable in secuencia:  
    sentencia
```

← *variable* va tomando el valor de cada elemento del iterable

Lazos (Loops)

```
while [condición] :  
    sentencias
```

```
for v in secuencia:  
    sentencias
```

```
# muestra nros. del 0 al 9  
a = 0  
while a < 10:  
    print(a)  
    a = a + 1
```

```
# muestra nros. del 0 al 9  
a = 0  
while True:  
    print(a)  
    a = a + 1  
    if a == 10:  
        break
```

Lazos (Loops)

```
while [condición] :  
    sentencias
```

```
for v in secuencia:  
    sentencias
```

```
# muestra nros. del 0 al 9  
for a in range(0,10):  
    print(a)
```

```
# muestra letra por letra  
for letra in "cadmio":  
    print(letra)
```

```
# muestra los nros en la lista  
for n in [1, 5, 8, 4]:  
    print(letra)
```

Lazos (Loops) - ejemplos

```
# sumatoria de los enteros
# entre 0 y 10
p = 0
suma = 0
while p <= 10:
    suma = suma + p
    p = p + 1

print(suma)
```

```
# múltiplos de 3 entre [n1..n2)
p = n1
while p < n2:
    if p % 3 == 0:
        print(p)
    p += 1

n = 10
while True:
    if n == 3:
        break
    n-=1
```

Lazos (Loops) - ejemplos

```
# cant de impares
# entre inicio y fin
inicio = int(input())
fin = int(input())
cant_impares = 0
p = inicio
while p <= fin:
    if p % 2 != 0:
        cant_impares += 1

    p = p + 1
```

```
# descendente entre 10 y 3
n = 10
while True:
    print(n)
    if n == 3:
        break
    n-=1
```

2. Secuencias y diccionarios

Strings

Strings

- Es un conjunto de caracteres entre ' ' o " " (Da igual)

```
frase = 'un objeto plateado'
```

- Para unir strings se usa + y para repetir *

```
"tu" * 2 + "ca"    # tutuca
```

- Se puede hacer obtener una parte del string (**slice**) con [índice] o [inicio:fin] (fin no inclusive)

```
frase[3:6]    # obj
```

Strings

- Para chequear si un string está o no dentro de otro string se puede usar `in`

```
texto = "una casa sobre una colina"
```

```
"sobre" in texto      # True
```

```
"UNA" in texto        # False
```

```
"kilo" not in texto   # True
```

Strings

- `len()` nos da la longitud del string: `len('el gato')`
- Para saber cuántas veces ocurre un string dentro de otro podemos usar `.count()`

```
'fue una muestra'.count("ue") # 2
```

- `.replace()` busca y reemplaza texto dentro de un string:

```
'fue una muestra'.replace('ue', '-?-') # 'f-?- una m-?-stra'
```

Strings

- Podemos **dividir un string** en varias strings con `split`

```
frase = 'un objeto plateado'
```

```
frase.split()      # ['un', 'objeto', 'plateado']
```

```
frase.split('e')   # ['un obj', 'to plat', 'ado']
```

- y muchas operaciones más... (startswith, endswith, find, upper, lower, isnumeric, isdecimal, etc.)

Strings

- Podemos **buscar** un string en otro con `find()`, `rfind()`

```
f = "La heladera"
f.find("hel")      # 3 posición donde encuentra 'hel'
f.rfind('e')      # 8 busca desde el fin del string
f.rfind('hila')   # -1 no encontró 'hila'
```

- `index()`, `rindex()` son similares pero si el string a buscar no se encuentra, lanzan `ValueError`

Strings

- Se puede crear un string con formato, a partir de literales y expresiones de diversos tipos (ej. números, strings):

```
'%d kilos de palta cuestan $%5.1f' % (4, 56.35)
```

Reemplaza `%d` por 4 y `%5.2f` por 56.35:

```
'4 kilos de palta cuestan $ 56.4'
```

- Antes del `%` se escribe un string con algunos especificadores de formato; Luego del `%` se indican literales y/o expresiones que van a reemplazar los especificadores de formato

Strings

- Cada especificador de formato se reemplaza, en orden, por una de las expresiones indicadas entre () a la derecha del %



```
frase = '%d %f' % (3, 9.57) # frase será '3 9.570000'
```

- Además del tipo se puede especificar la cantidad de caracteres (como mínimo) que va a ocupar la expresión.
- Y en el caso de números con decimales, cuántos números se mostrarán luego del punto.

```
frase = '%3d %4.1f' % (3, 9.57) # frase será ' 3  9.6'
```

Strings

Hay muchos especificadores de formato... algunos comunes:

- `%d`, `%i` número entero

`'%d + %d = %3d' % (4, 5, 4 + 5)` → `'4 + 5 = 9'`

`%+4d` usa seis lugares, muestra el signo + o -

`%-3d` entero, tres lugares, alineado a izquierda

`'%-6d y %+6d' % (100, 200)` → `'100 y +200'`

`'%+05d' % (3)` → `'+0003'`

Strings

- `%s` string
- `%f` número (punto flotante)
`%.2f` con dos decimales
`%6.2f` total de seis lugares de ancho (con decimales y .)
`'Nota: %3.1f - estado: %s' % (6.39, "Aprobado")` →
`'Nota: 6.4 - estado: Aprobado'`
- `%e`, `%E` muestra un número en notación científica
`'%.2e - %+.1E' % (0.0156, 0.0156)` → `'1.56e-02 - +1.6E-02'`

Strings

- `%x`, `%H` muestra un entero en hexadecimal

`'0x%02X en hexadecimal es %d en decimal' % (95, 95) →`
`'0x5F en hexadecimal es 95 en decimal'`

- `%c` un solo caracter

`"El %d es el código ASCII de '%c'" % (65, 65) →`
`"El 65 es el código ASCII de 'A'"`

Strings

- Si queremos usar un string que va a necesitar varias líneas podemos indicarla con tres comillas (dobles o simples)

```
texto = '''la casa es  
         grande y fresca, muy adecuada para los  
         veranos de la zona'''
```

- Los strings indicados entre ''' conservan los saltos de línea, tabulaciones y caracteres especiales (Ej. \)

Listas

Listas

- Son conjuntos de ítems -variables, objetos, etc.- y se escriben entre [y]

```
listaVacia = []  
listaDePares = [2, 4, 6, 8, 10]  
listaTexto = ["a"]
```

- Los elementos de la lista pueden ser de diferentes tipos:

```
unaLista = [0, "a", "poroto", 7.65]
```

Listas

- A cada **ítem** le corresponde un número de orden: su **índice**
- Con este índice se puede acceder al valor del ítem correspondiente o modificarlo.

```
>>> laLista = [8, 32, 16, 2]

>>> print(laLista[1])      # muestra 32

>>> laLista[2] = 64

>>> print(laLista[2])      # muestra 64
```

Listas

- Para agregar ítems se usa el método `append()`

```
mascotas = ['Pata', 'Poroto']  
mascotas.append('Chuleta')  
print(mascotas)
```

```
['Pata', 'Poroto', 'Chuleta']
```

- Para insertar en una posición determinada:

```
mascotas.insert(1, 'Puré')
```

```
['Pata', 'Puré', 'Poroto', 'Chuleta']
```

Listas

- Para borrar items se puede usar `del()` o `remove()`

```
compras = ['agua', 'verdura', 'fruta']  
del(compras[1])    ← borra el elemento con índice 1
```

0...

```
compras.remove('verdura') ← o borra 'verduras'
```

Ambos borran 'verduras'...

Listas

- Algunas operaciones más:

```
llegada = ['Ana', 'Juan', 'Otto', 'Ana', 'Marta']
```

```
len(llegada)          # longitud
```

```
llegada.count('Ana')  # cantidad de veces que aparece un ítem
```

```
llegada.reverse()
```

```
llegada.index('Otto') # primer índice donde aparece el ítem
```

Listas

- Para ordenar listas:

```
puntos = [49, 87, 67, 98, 32, 76]
```

```
puntosOrdenados = sorted(puntos) # no modifica lista original
```

```
puntos.sort() # modifica la lista original
```

Tuplas

Tuplas

- Al igual que las listas, permiten agrupar elemento pero son **Inmutables**.
- Se indican entre (y)

```
ingreso = (20, 6, 1990)
```

```
tupla = (8, )    # un valor... si se escribe (8) es un entero,  
                # no una tupla!!
```

```
u = ingreso[1]   # asigna a u el elemento 1 (6 en este ejemplo)
```

```
ingreso[1] = 7   # error!!! son inmutables!!
```

Empaquetado de tuplas...

- Empaquetado:

Se crea una tupla a partir de varias variables:

```
d = 15
m = 6
a = 1998
fnac = d, m, a      # fnac = (15, 6, 1998)

nombre = 'Noel'
edad = 67
persona = nombre, edad, fnac  # ('Noel', 67, (15, 6, 1998))
```

Desempaquetado de tuplas y secuencias...

- Se asignan los elementos de una secuencia a varias variables:

```
p = (9, 3, -7)
x, y, z = p          # x = 9  y = 3  z = -7

u, v = p             # ERROR... (falta una variable)

u, v, v = p          # u = 9  v = -7

a, b, c = "Por"       # funciona con strings: a='P' b='o' c='r'

g, h = ['uno', 'dos'] # ...y con listas y strings (y con
                      # secuencias en general)
```

Desempaquetado de secuencias...

- Se asignan los elementos de una secuencia a varias variables:

```
p = (9, 3, -7, 8)
x, *y = p           # x = 9   y = (3, -7, 8)

q = [10, 9, 7, 4]
x, y, *z = q        # x = 10  y = 9   z = [7, 4]

a, b, *c = "camas"  # a = 'c'  b = 'a'  c = ['m', 'a', 's']
```

El `*` antes de una variable indica que en esa variable va "lo que no se usó de la secuencia"

Operaciones con secuencias

- strings, listas, tuplas son parte del conjunto **secuencias**
- Tienen operaciones en común:

<code>x in s</code>	indica si x está en s (devuelve True o False)
<code>len(s)</code>	cantidad de elementos de s
<code>max(s)</code>	máximo elemento de s
<code>min(s)</code>	mínimo elemento de s
<code>s + t</code>	concatena s y t
<code>s[i]</code>	elemento i de s
<code>s[i:j]</code>	elementos de s desde i (inclusive) hasta j (no inclusive)
<code>s[i:j:k]</code>	elementos de s desde i hasta j con "paso" k

Operaciones con secuencias - Slicing

- Se pueden especificar segmentos de strings, listas, tuplas usando **slices** especificando entre `[` y `]` el índice inicial (inclusive) y el final (exclusive) separados por `:`

```
pares = [ 2, 4, 6, 8, 10, 12, 14, 16]
```

```
pares[2:5]      # devuelve [6, 8, 10]  
pares[5:]       # devuelve [12, 14, 16]  
pares[:3]       # devuelve [2, 4, 6]  
pares[::3]      # [2, 8, 14]
```

Operaciones con secuencias - Slicing (cont.)

- Se pueden usar números negativos: se cuenta desde el final.

```
s = "abcde"
s[-2]           # devuelve d
s[-4:-2]        # devuelve bc
s[-4:4]         # devuelve bcd
s[:-2]          # devuelve abc
s[::-2]         # devuelve eca

(1, 2, 5, 9)[:2:-1]  # (9,)
(1, 2, 5, 9)[: -1:]  # (1, 2, 5)
```

Diccionarios

Diccionarios

- Es un conjunto de pares, cada uno compuesto por una clave ("*key*") y un valor ("*value*") separados por **:**
- Se indican entre **{** y **}**

```
puntaje = {'Carla': 7, 'Maxi':2, 'Rita': 8}
```

- Para acceder a cada valor se usa la clave entre **[]** o **.get()**

```
puntaje['Maxi']      # devuelve 2
```

Diccionarios

- Para acceder a un valor se indica la clave asociada entre [] o .get()

```
Si      d = {'manzana':67.4, 'pera':45.0}

d['manzana']    o d.get('manzana') # ambos devuelven 67.4
d['limon']      # si no está la clave da excepción: KeyError!!!!
d.get('limon')  # .get() no genera excepción
                si no existe la clave; devuelve None...

d.get('limon', 'NO') # ...o el valor indicado a continuación
                    de la clave; en este ejemplo devuelve NO
```

Diccionarios

- Agregamos o modificamos elementos a un diccionario con:
`nombrediccionario[clave] = valor`
- Si clave existe, modifica el valor.
- Si no existe, crea el elemento con clave:valor especificados.

```
mascotas = {'Tango': 'gato', 'Pata': 'perro'}  
mascotas['Pata'] = 'perra'           # modifica  
mascotas['Poroto'] = 'perro'         # agrega  
  
{ 'Poroto': 'perro', 'Pata': 'perra', 'Tango': 'gato' }
```

Diccionarios

- Se pueden borrar elementos con `del()`

```
letras = {'a': 7, 't': 5, 'u': 8}

del letras['t']      # {'a': 7, 'u': 8}
```

- `.values()` y `.keys()` devuelven lista con valores y claves respectivamente.
- `n in d.values()` devuelve True si el valor n existe en el diccionario d

3. Funciones y módulos

Funciones

- Para crear "funciones" usamos def

```
def elevar(base, exponente):  
    resultado = base ** exponente  
    return resultado  
  
print(elevar(2, 4))    # 16
```

- Con `return` se indica lo que devuelve la función
- Entre paréntesis se indican los parámetros.
- En una función el `return` y los parámetros son opcionales

Funciones

- En una función el `return` y los parámetros son opcionales

```
def mostrarEncabezado():  
    print("=" * 32)  
    print(" PID           %CPU           RAM")  
    print("=" * 32)  
  
mostrarEncabezado()
```

Funciones

- Una función puede tener parámetros con valores por defecto

```
def sumar(a, b = 9):  
    return a + b  
  
print(sumar(8, 4))      # muestra 12  
print(sumar(8))         # muestra 17
```

- Los argumentos son los valores que le damos a los parámetros al llamar a la función.
- En el ejemplo: a y b son parámetros. En el primer caso se llama a "sumar" con argumentos 8 y 4.

Módulos (Bibliotecas)

- Para utilizar un módulo se hace `import` módulo
- Ej.

```
import math
```

- Otras bibliotecas "comunes":

math, datetime, time, random, os, sys, posix, socket,
unittest, gettext, locale, argparse, numpy, sqlite3, matplotlib...

<https://pypi.org/>

<https://docs.python.org/3/py-modindex.html>

<https://pythontips.com/2013/07/30/20-python-libraries-you-cant-live-without/>

Módulos (Bibliotecas)

- Se pueden utilizar módulos propios.
- Se puede "importar" cualquier archivo en Python.

enfaticar.py

```
def exclamar(s, n):  
    print('¡' * n, end='')  
    print(s, end='')  
    print('!' * n)  
  
def preguntar(s, n):  
    print('¿' * n, end='')  
    print(s, end='')  
    print('? * n)
```

"incorpora" a enfaticar.py



```
import enfaticar
```

```
enfaticar.exclamar("Epa", 2)  
enfaticar.preguntar("Una", 4)
```



ejecuta funciones de "enfaticar"

Módulos (Bibliotecas)

enfaticar.py

```
def exclamar(s, n):  
    print('¡' * n, end='')  
    print(s, end='')  
    print('!' * n)  
  
def preguntar(s, n):  
    print('¿' * n, end='')  
    print(s, end='')  
    print('? ' * n)  
  
if __name__ == '__main__':  
    print("pruebo enfaticar")
```

- Otras opciones:

```
import enfaticar as enf  
  
enf.exclamar("Epa", 2)  
enf.preguntar("Una", 4)
```

```
from enfaticar import *  
  
exclamar("Epa", 2)  
preguntar("Una", 4)
```

Así, si se hace `import enfaticar`, el print no se ejecuta

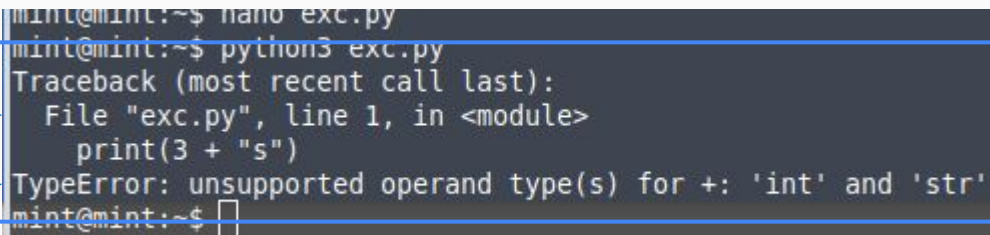
4. Excepciones

Excepciones

- Una excepción es un evento que se produce cuando algo sale mal ("error") mientras se ejecuta el programa (runtime error)

Línea que generó la excepción

Qué excepción ocurrió



```
mint@mint:~$ nano exc.py
mint@mint:~$ python3 exc.py
Traceback (most recent call last):
  File "exc.py", line 1, in <module>
    print(3 + "s")
TypeError: unsupported operand type(s) for +: 'int' and 'str'
mint@mint:~$
```

- Son una forma de enviar mensajes entre objetos...
- Si no son deseados, hay que evitar que el usuario vea estos errores.

Excepciones

- Para "atajar" los errores que se producen en tiempo de ejecución se usa la estructura

```
try:  
    ...  
except:  
    ...
```

`else` y `finally`,
son opcionales

```
try:  
    código que puede fallar  
    .....  
except Exception1:  
    si se produce Exception1  
except Exception2:  
    si se produce Exception2  
else:  
    si no hubo excepciones  
  
finally:  
    siempre, al final
```



Excepciones

```
try:
    a = int(input('Ingrese N1: '))
    b = int(input('Ingrese N2: '))
    resultado = a / b
except ZeroDivisionError:
    print('N2 no puede ser cero')
except ValueError:
    print('N1 y N2 deben ser números enteros')
except:
    print('Error en la operación')
else:
    print('%d / %d = %.2f' % (a, b, resultado))
```

5. Archivos

Archivos de texto

- Para utilizar un archivo hay que:
 - Abrirlo
 - Operar con su contenido (opcional... !?)
 - Cerrarlo
- Al abrir el archivo recibimos del sistema operativo un identificador que refiere al fichero especificado.
- Se puede abrir para escribir (w) leer (r) o agregar (a)
- Si se abre como para escribir (w) o agregar (a) y no existe, se crea un archivo vacío. Si se abre para lectura (r) y no existe se genera error

Archivos de texto

- Para abrir un archivo se usa **open**:

```
f = open("prueba.txt", "w") # para escribir
```

nombre del archivo

modo

```
f = open("prueba.txt", "r") # para leer
```

```
f = open("prueba.txt", "a") # para agregar
```

Archivos de texto

- Una vez abierto el archivo podemos escribir con **write**

```
f.write("primera línea\n")    # escribe  
f.write("segunda línea\n")    # escribe
```



\n para que "baje" a otra línea

Archivos de texto

- Para leer podemos usar `readline` o `readlines`

```
f.readline()    # devuelve un string con una línea del  
                # archivo.
```

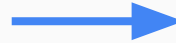
Si se intenta leer al final del archivo devuelve ""

```
f.readlines()  # devuelve una lista con todas las líneas  
                # del archivo
```

Archivos de texto

- Un ejemplo de escritura:

```
f = open("prueba.txt", "w")
f.write("valores\n")
f.write("=" * 7 + "\n")
for n in range(1, 10):
    f.write("%03d\n" % (n))
f.close()
```



```
mint@mint:~$ cat prueba.txt
valores
=====
001
002
003 do en Drive
004
005
006
007
008
009
mint@mint:~$
```

Archivos de texto

- Un ejemplo de lectura de un archivo:

```
f = open("prueba.txt", "r")  # abre para lectura
s = f.readline()             # lee primera línea (si hay)
while s != "":               # si no es fin de archivo...
    print(s)                  # muestra línea leída
    s = f.readline()          # lee siguiente línea
f.close()                     # cierra archivo
```

Archivos de texto

- Otro ejemplo de lectura (más compacto)

```
f = open("prueba.txt", "r")
for s in f:    # se puede iterar
    print(s)
f.close()
```

6. JSON

JSON

- JSON (JavaScript Object Notation) es un formato de intercambio de datos muy utilizado.
- Basado en texto.
- Estructurado.

Ej.

```
{  
  "name": "John",  
  "age": 30,  
  "city": "New York"  
}
```

JSON

Otro ejemplo:

```
{
  "nombre": "Alfredo",
  "apellido": "Zitarrosa",
  "genero": ["zamba", "milonga", "candombe"],
  "nacio": 1936,
  "discos": [
    {
      "titulo": "Yo sé quién soy",
      "año": 1968
    },
    {
      "titulo": "Adagio en mi país",
      "año": 1973
    }
  ]
}
```

- Además de strings, números y booleanos se puede representar listas.
- También otros objetos anidados.

JSON

- Python cuenta con un módulo (`"json"`) que simplifica la tarea de exportar datos a formato JSON y de interpretar JSON como formatos de Python.
- `json.loads(s)` devuelve un diccionario a partir de un JSON en la variable `s`
- `json.dumps(d)` devuelve un string con formato JSON a partir de un diccionario `d`

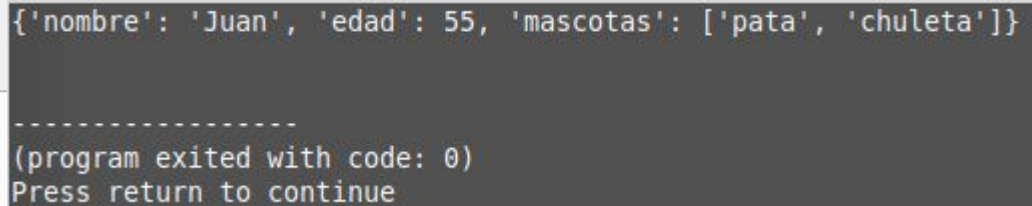
JSON

```
# ejemplo: convierte string JSON a diccionario Python
import json

jsonTest = '{"nombre": "Juan",
            "edad": 55,
            "mascotas":["pata", "chuleta"]}'

s = json.loads(jsonTest)

print(s)
```



A screenshot of a terminal window. The top bar shows menu options: File, Edit, View, Search, Terminal, Help. The main content area displays the output of the Python script: a JSON object converted to a Python dictionary: {'nombre': 'Juan', 'edad': 55, 'mascotas': ['pata', 'chuleta']}. Below this, a separator line of dashes is shown, followed by the message '(program exited with code: 0)' and 'Press return to continue'. A cursor is visible at the bottom left.

```
{'nombre': 'Juan', 'edad': 55, 'mascotas': ['pata', 'chuleta']}
```

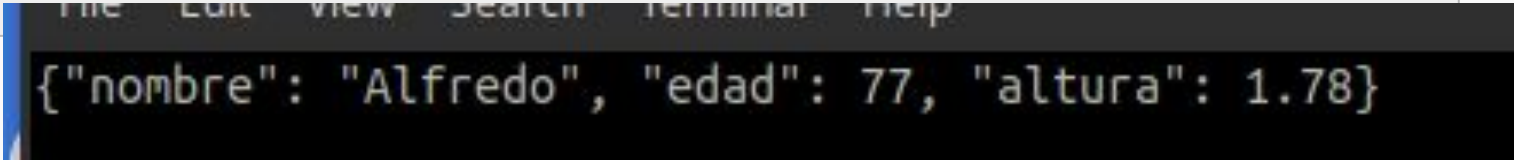
```
-----
(program exited with code: 0)
Press return to continue
█
```

JSON

```
# ejemplo: convierte diccionario Python a string JSON
import json

p = {"nombre": "Alfredo", "edad": 77, "altura": 1.78}
s = json.dumps(p)

print(s)
```



The screenshot shows a terminal window with a dark background and light-colored text. The output of the code is displayed as a single line: {"nombre": "Alfredo", "edad": 77, "altura": 1.78}. The terminal has a menu bar at the top with options like File, Edit, View, Search, Terminal, and Help.

```
{"nombre": "Alfredo", "edad": 77, "altura": 1.78}
```

JSON

- En los ejemplos anteriores se trabaja con estructuras (JSON y diccionarios) en memoria.
- Es habitual que los JSON se encuentren en archivos en disco o en servidores remotos.

JSON

```
# ejemplo: lee JSON y lo transforma a diccionario Python
import json

# colors.json contiene datos en formato JSON
f = open('colors.json', 'r')

d = json.loads(f.read())

print(d)
```

JSON

```
# ejemplo: genera archivo JSON a partir de un diccionario
import json

cli = {'zona':4, 'nombres':['Alvarez', 'Perez']}

# clientes.json contiene datos en formato JSON
f = open('clientes.json', 'w')

f.write(json.dumps(cli))

f.close()
```

JSON

```
# ejemplo: lee JSON desde internet
import json
import requests

url = "http://api.open-notify.org/iss-now.json"
jsonPosi = requests.get(url)
p = json.loads(jsonPosi.text)

print(p)
```

7. Funciones anónimas

map / filter

comprensión de listas

Lambda

- Son funciones "sin nombre" y de "un solo uso"
- Se usan cuando hay que pasar una función como parámetro.

lambda argumentos: expresión

Ej.:

```
print((lambda p, q: p * q)(4, 6))           o...
```

```
f = lambda p, q: p * q  
print(f(4, 6))
```

map

`map(función, iterable1, iterable2, ..., iterableN)`

Devuelve un iterable resultante de aplicar *función* al/los iterable/s proporcionados como argumentos.

Ej.

```
lista = [3, 6, 7, 3, 8]
print(list(map(lambda s: s * 3, lista)))

# muestra [9, 18, 21, 9, 24]
```

filter

`filter( función, iterable)`

Devuelve un iterable con los elementos del iterable proporcionado como parámetro, para los que función sea True

Ej.

```
lista = [3, 6, 7, 3, 8]
print(list(filter(lambda s: s % 2 == 0, lista)))

muestra [6, 8]
```

8. SQLite

SQLite

- Se puede trabajar con bases de datos SQLite con el módulo `sqlite3`; se incluye con Python desde la versión 2.5 en adelante.

```
import sqlite3
conn = sqlite3.connect('test.db') # conecta con la base
                                   test.db. Si no existe,
                                   la crea

conn.close()                      # cierra conexión
```

- GUI para trabajar en modo visual con SQLite: <https://sqlitebrowser.org/>

SQLite

- Con `.execute` se ejecutan sentencias SQL

```
# crea tabla mascotas
conn.execute('''CREATE TABLE mascotas
              (id          INT PRIMARY KEY NOT NULL,
               nombre     TEXT           NOT NULL,
               edad       INT           NOT NULL);''')
```

- Si la tabla existe, genera una excepción
- Desde SQLite 3.3 se puede usar:

```
CREATE TABLE IF NOT EXISTS mascotas
```

SQLite

- Con `.execute` se ejecutan sentencias SQL

```
# agrega datos en mascotas
```

```
conn.execute("INSERT INTO mascotas (id, nombre, edad) \n              VALUES (1, 'Pata', 9)");
```

```
conn.execute("INSERT INTO mascotas (id, nombre, edad) \n              VALUES (2, 'Poroto', 13)");
```

```
conn.commit()      # hace efectivas las consultas previas
```

SQLite

- Con `.execute` se ejecutan sentencias SQL

```
# borra datos en mascotas
conn.execute("DELETE FROM mascotas WHERE id = 1");

conn.commit()      # hace efectivas las consultas previas

print("Se borraron %d filas" % conn.total_changes)
```

SQLite

- Con `.execute` se ejecutan sentencias SQL

```
# borra datos en mascotas
conn.execute("UPDATE mascotas SET edad = 10 WHERE id = 1");

conn.commit()      # hace efectivas las consultas previas

print("Se modificaron %d filas" % conn.total_changes)
```

SQLite

- Ejecutar un SELECT devuelve un `cursor` que nos facilita trabajar con los resultados obtenidos

```
crs = conn.execute("SELECT id, nombre, edad FROM mascotas")

print("id\tnombre\t\t\tedad\t")
for fila in crs:
    print("%3d\t%s\t\t\t%2d" % \
          (fila[0], \
            fila[1], \
            fila[2]))
```

9. GUI

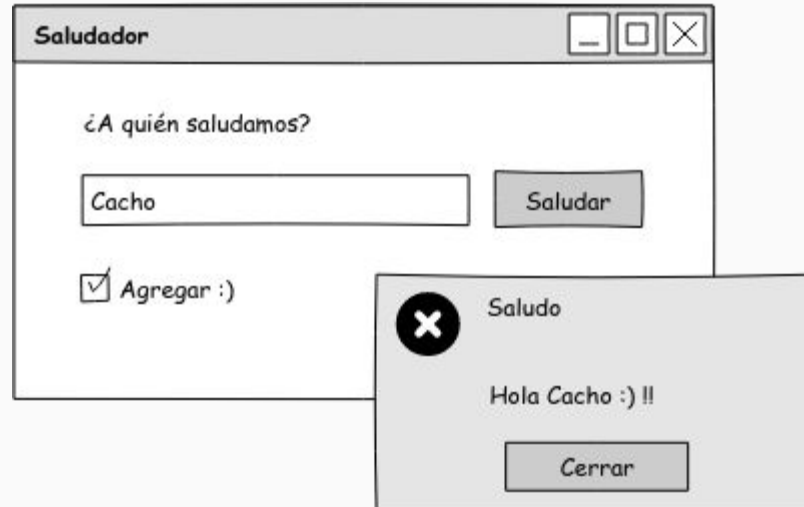
GUI

Algunas opciones:

Toolkits para interfaces gráficas de usuario

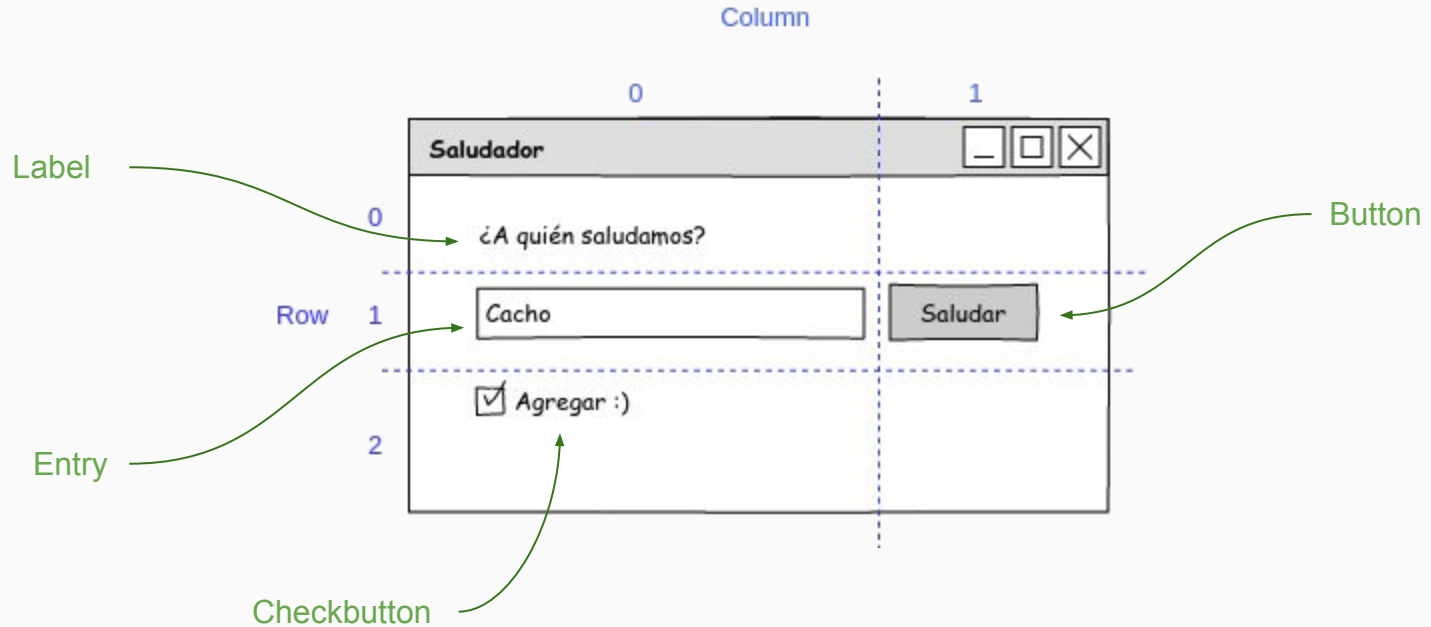
- Tk (viene con Python*)
- wxWidgets
- Qt
- GTK
- Kivy

GUI



Estos ejemplos en: <https://github.com/diegocode/python-gui-examples>

GUI - Tk



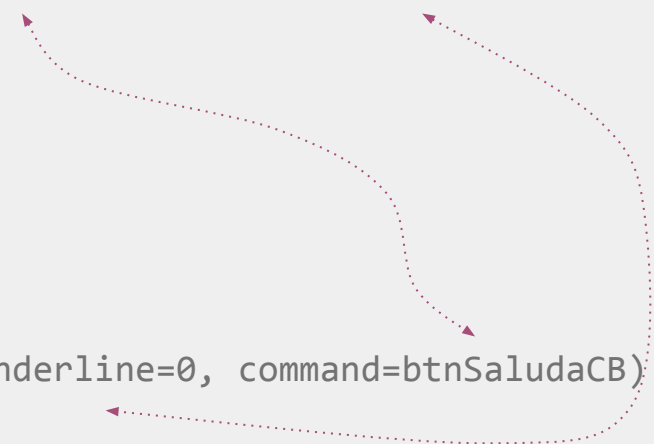
GUI - Tk

```
import tkinter
from tkinter import messagebox

top = tkinter.Tk()      # ventana principal
top.title("Saludador")  # título de la ventana principal

# esta función se ejecuta cuando se presiona el botón (o cuando presionan Alt + s)
def btnSaludaCB():
    s = "Hola " + txtNombre.get()
    if flgSonrisa.get() == 1:
        s = s + " :)"
    s = s + "!!!"
    messagebox.showinfo("Saludo", s)

# botón
btnSaludar = tkinter.Button(top, text="Saludar", underline=0, command=btnSaludaCB)
top.bind('<Alt_L><s>', lambda e:btnSaludarCB())
```



The diagram consists of three red dotted arrows. The first arrow starts from the `command=btnSaludaCB` parameter in the `tkinter.Button` constructor and points to the `def btnSaludaCB():` function definition. The second arrow starts from the `lambda e:btnSaludarCB()` argument in the `top.bind` call and also points to the `def btnSaludaCB():` function definition. The third arrow starts from the `btnSaludarCB()` call inside the `lambda` expression and points to the `def btnSaludaCB():` function definition.

GUI - Tk

```
# texto estático (Label)
lblAQuien = tkinter.Label(top, text = "¿A quién saludamos?")

# entrada de texto (Entry)
txtNombre = tkinter.Entry(top)

# checkbox
flgSonrisa = tkinter.IntVar()          # variable para saber si está chequeado o no
chkSonrisa = tkinter.Checkbutton(top, text = "Agregar :)", \
                                   variable = flgSonrisa, height = 3)

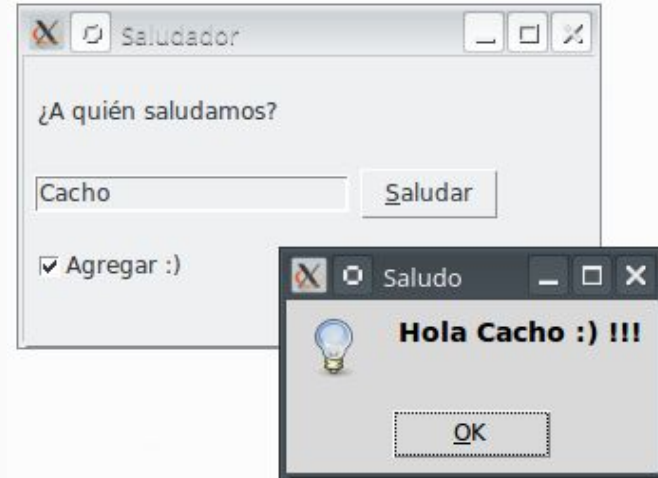
# posiciona cada widget en columna y fila correspondiente
lblAQuien.grid(row=0, column=0, sticky=tkinter.W, columnspan=2, padx=5, pady=20)
txtNombre.grid(row=1, column=0, sticky=tkinter.W, padx=5)
btnSaludar.grid(row=1, column=1)
chkSonrisa.grid(row=2, column=0, sticky=tkinter.W, columnspan=2)

top.mainloop()      # queda esperando eventos
```

GUI - Tk



Windows 7



Linux Mint 18.3 (KDE 5)

GUI - wxWidgets

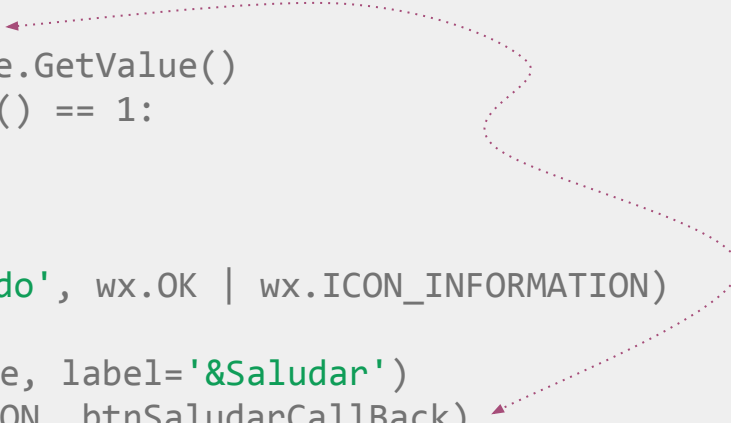
```
import wx

app = wx.App()
frame = wx.Frame(None, title='Saludador')

def btnSaludarCallBack(e):
    s = "Hola " + txtNombre.GetValue()
    if chkSonrisa.GetValue() == 1:
        s = s + " :) "

    s = s + "!!!"
    wx.MessageBox(s, 'Saludo', wx.OK | wx.ICON_INFORMATION)

btnSaludar = wx.Button(frame, label='&Saludar')
btnSaludar.Bind(wx.EVT_BUTTON, btnSaludarCallBack)
```

A diagram consisting of two red dotted lines with arrowheads. The first line starts from the 'e' parameter in the function definition 'def btnSaludarCallBack(e):' and points to the 'e' argument in the 'Bind' method call 'btnSaludar.Bind(wx.EVT_BUTTON, btnSaludarCallBack)'. The second line starts from the 'btnSaludarCallBack' function name in the 'Bind' method call and points to the same 'e' argument, indicating the binding of the event to the function.

GUI - wxWidgets

```
lblAQuien = wx.StaticText(frame, label=u"¿A quién saludamos?")
txtNombre = wx.TextCtrl(frame)
chkSonrisa = wx.CheckBox(frame, label='Agregar :)')

gs = wx.GridBagSizer(5, 5)
gs.Add(lblAQuien, pos=(0, 0), flag= wx.EXPAND | wx.ALL, border=5)
gs.Add(txtNombre, pos=(1, 0), flag= wx.EXPAND | wx.ALL, border=5)
gs.Add(btnSaludar, pos=(1, 1), flag= wx.EXPAND | wx.ALL, border=5)
gs.Add(chkSonrisa, pos=(2, 0), flag= wx.EXPAND | wx.ALL, border=5)

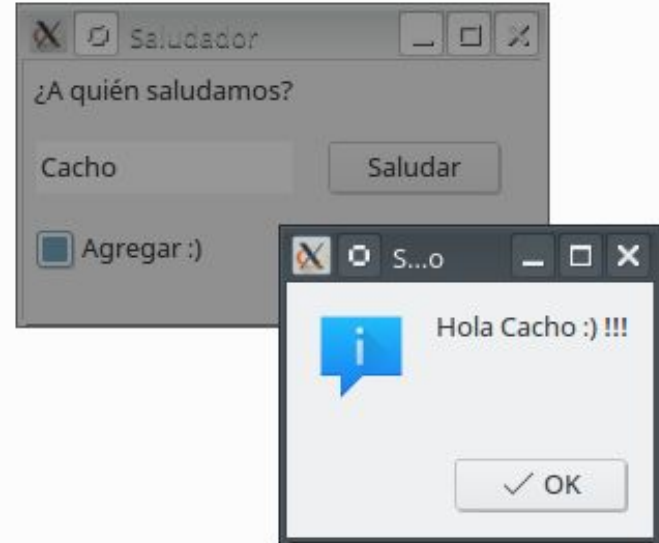
frame.SetSizer(gs)

frame.Show()
frame.Centre()
app.MainLoop()
```

GUI - wxWidgets



Windows 7



Linux Mint 18.3 (KDE 5)

GUI - Qt

```
# -*- coding: utf-8 -*-

import sys
from PyQt4 import QtGui

app = QtGui.QApplication(sys.argv)

w = QtGui.QWidget()
w.setGeometry(100, 100, 200, 100) # x, y, ancho, alto

txtNombre = QtGui.QLineEdit(w)
chkSonrisa = QtGui.QCheckBox(w)
chkSonrisa.setText("Agregar :)")

lblAQuien = QtGui.QLabel(w)
lblAQuien.setText(u"¿A quién saludamos?")
```

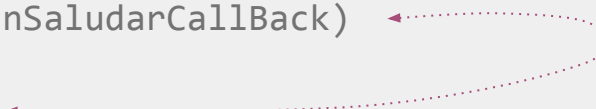
GUI - Qt

```
btnSaludar = QtGui.QPushButton(w)
btnSaludar.setText("&Saludar")
btnSaludar.clicked.connect(btnSaludarCallBack)

def btnSaludarCallBack():
    s = "Hola " + txtNombre.text()
    if chkSonrisa.isChecked():
        s = s + " :)"

    s = s + "!!!"

    msg = QtGui.QMessageBox()
    msg.setIcon(QtGui.QMessageBox.Information)
    msg.setText(s)
    msg.setWindowTitle("Saludo")
    msg.setStandardButtons(QtGui.QMessageBox.Ok)
    msg.exec_()
```



GUI - Qt

```
grid = QtGui.QGridLayout()
grid.addWidget(lblAQuien, 0, 0)
grid.addWidget(txtNombre, 1, 0)
grid.addWidget(chkSonrisa, 2, 0)
grid.addWidget(btnSaludar, 1, 1)

w.setWindowTitle("Saludador")
w.setLayout(grid)

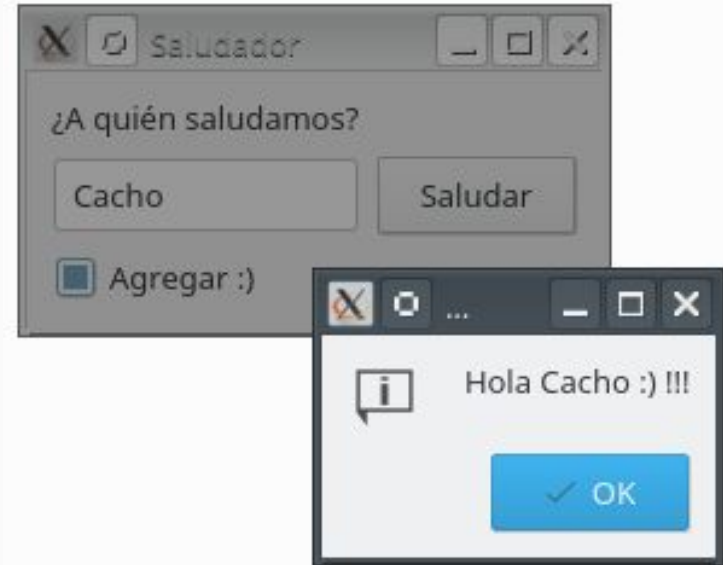
w.show()

sys.exit(app.exec_())
```

GUI - Qt



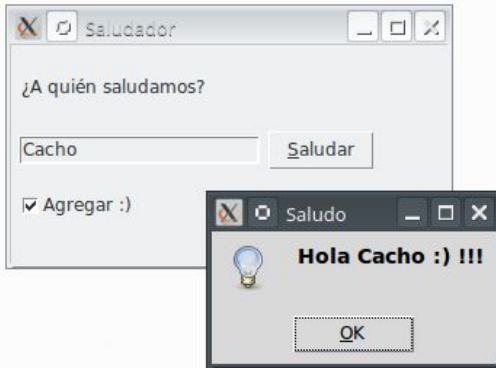
Windows 7



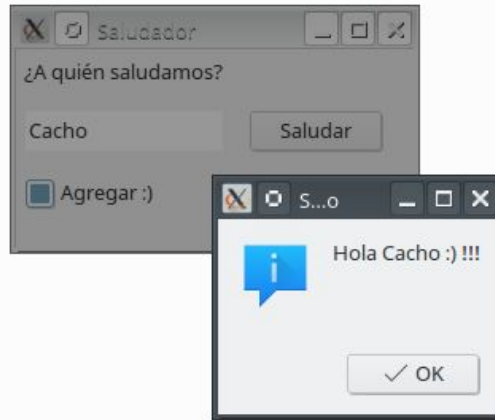
Linux Mint 18.3 (KDE 5)

GUI

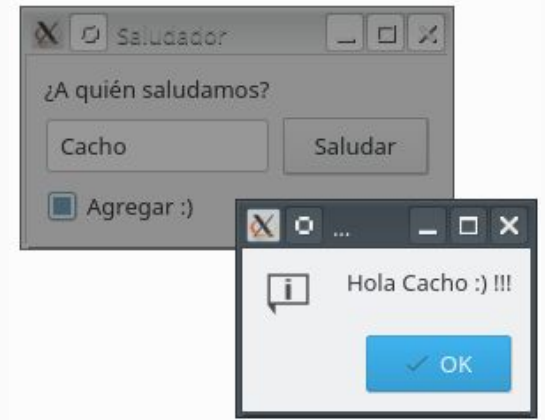
Linux Mint 18.3 (KDE 5)



Tk



wxWidgets



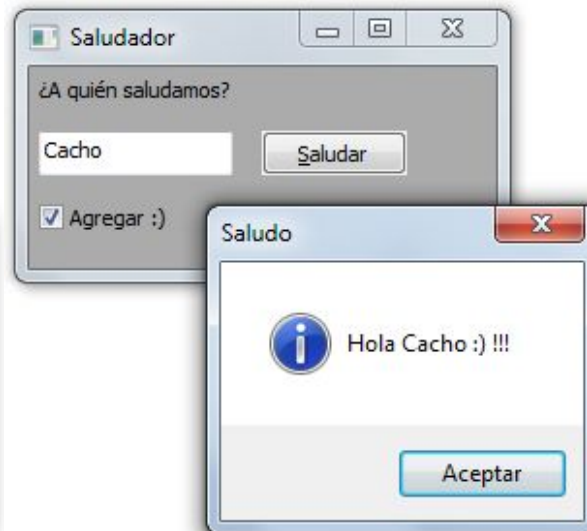
Qt

GUI

Windows 7



Tk



wxWidgets



Qt

10. Frameworks Web

Framework web

- Un framework web es un conjunto de módulos, etc. que permiten implementar un site web. En general permiten realizar al menos estas tareas
 - Ruteado de URL (URL routing) a una sección de código
 - Soporte de objetos Request y Response que encapsulan la información a y desde el browser.
 - Motor de Templates (Template engine) que permite implementar código "mezclado" con el HTML
 - Servidor Web de desarrollo

Framework web

- Algunas opciones:

- Django
- Flask
- Falcon
- Tornado
- Pyramid
- Web2Py
- Bottle

etc.

Bottle

- Es un framework web completo en un único módulo, sin dependencias.

<https://bottlepy.org>

En el único archivo que se necesita contiene

- Routing
- Manejo de Request / Response
- Template engine
- Web Server

Bottle

- Para comenzar a usarlo se instala (ej. se baja) **bottle.py** en el directorio de nuestro proyecto:

```
wget http://bottlepy.org/bottle.py
```

- Para testear si funciona:

```
from bottle import route, run

@route('/hello')
def hello():
    return "Hello World!"

run(host='localhost', port=8080, debug=True)
```



En un archivo
pruebaBottle.py

- Lo ejecutamos: `python3 pruebaBottle.py`
- Con un browser accedemos a <http://localhost:8080/hello>

Bottle

Cada vez que se accede a
/hello se ejecuta
def saludar():

Lo que "devuelve" se
muestra en el browser

Inicia el servidor web.
"Escucha" en el port 8080 de
localhost (127.0.0.1)
Si cambia el código fuente, lo
reinicia automáticamente.

```
from bottle import route, run
```

```
@route('/hello')
```

```
def saludar():
```

```
    return "Hola!!"
```

```
run(host='localhost', \
    port=8080, \
    debug=True, \
    reloader=True)
```

Bottle

Se pueden asignar varias
URL a un método

Reemplaza {{nombre}} por
inpNombre

localhost:8080/hello/Cacho

Va a mostrar:

Hola Cacho!!!

```
from bottle import route, run, template

@route('/')
@route('/hello/<inpNombre>')
def saludar(inpNombre = 'Capo'):
    return template('Hola <b>{{nombre}}!!!</b>',
                    nombre = inpNombre)

run(host='localhost',
    port=8080,
    debug=True,
    reloader=True)
```

Bottle

```
@route('/tabla/<num>')
def muestraTabla(num):
    return template('muestra_tabla', numero = int(num))
```

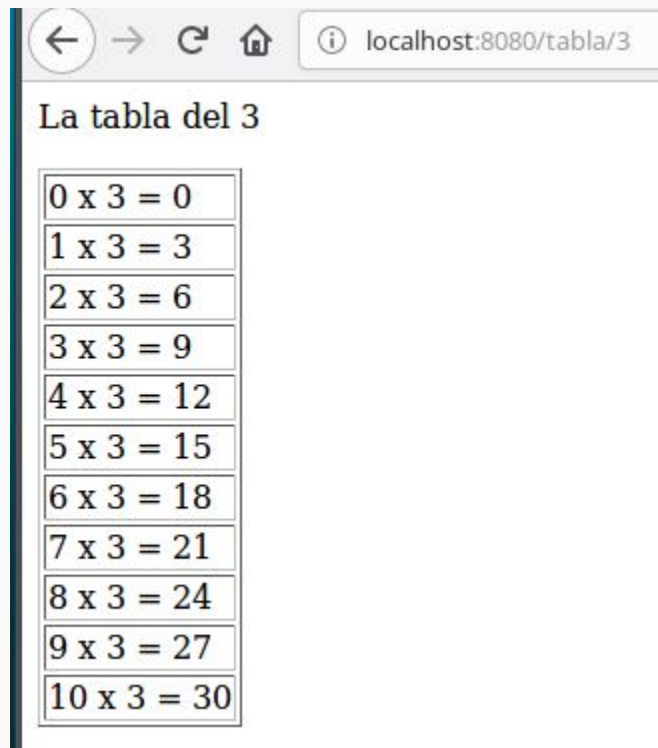
Se va a mostrar el
template
muestra_tabla

muestra_tabla.tpl

```
%#template para mostrar tabla de multiplicar
<p>La tabla del {{numero}}</p>
<table border="1">
%n = 0
%while n <= 10:
    <tr>
        <td>{{n}} x {{numero}} = {{n * numero}}</td>
    </tr>
    %n = n + 1
%end
</table>
```

- Se puede mezclar código Python con el HTML
- Se comienza con % cada línea de Python
- En lugar de indentar para agrupar sentencias, se usa %end para cerrar while, if, etc.

Bottle



A screenshot of a web browser window. The address bar shows 'localhost:8080/tabla/3'. The page title is 'La tabla del 3'. The content is a vertical list of 11 multiplication problems, each in its own box: 0 x 3 = 0, 1 x 3 = 3, 2 x 3 = 6, 3 x 3 = 9, 4 x 3 = 12, 5 x 3 = 15, 6 x 3 = 18, 7 x 3 = 21, 8 x 3 = 24, 9 x 3 = 27, and 10 x 3 = 30.

La tabla del 3	
0	0 x 3 = 0
1	1 x 3 = 3
2	2 x 3 = 6
3	3 x 3 = 9
4	4 x 3 = 12
5	5 x 3 = 15
6	6 x 3 = 18
7	7 x 3 = 21
8	8 x 3 = 24
9	9 x 3 = 27
10	10 x 3 = 30

Bottle

```
from bottle import route, run, template, request

@route('/')
def inicio():
    return template('form')

@route('/tabla')
def muestraTabla():
    num = request.GET.get('tabladel', '').strip()
    return template('muestra_tabla', numero = int(num))
```

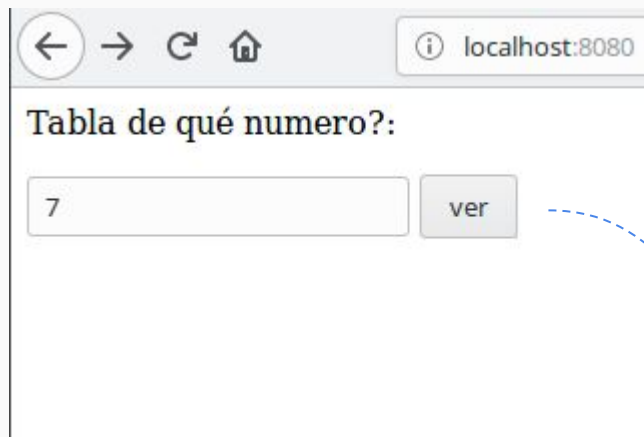
Para usar un formulario

El mismo template del ejemplo anterior.

form.tpl

```
<p>Tabla de qué numero?:</p>
<form action="/tabla" method="GET">
    <input type="text" name="tabladel">
    <input type="submit" value="ver">
</form>
```

Bottle



← → ↻ 🏠 localhost:8080

Tabla de qué numero?:



← → ↻ 🏠 localhost:8080/tabladel=7

La tabla del 7

0 x 7 = 0
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70

Recursos

Bibliografía & recursos

Documentación Python

<https://docs.python.org/3/>

Apunte UBA

<http://materias.fi.uba.ar/7501/apunte%20PYTHON.pdf>

Python para todos

http://www.utic.edu.py/citil/images/Manuales/Python_para_todos.pdf