

# Ejercicios de Python I

## El modo interactivo.

Para entrar en el modo interactivo de *Python* ejecutar la orden *python* en una terminal. Hacer los siguientes ejercicios:

1. Comprobar qué versión del Python estamos utilizando.

```
C:\Python27\python.exe
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2. Probar el modo calculadora. Operaciones con +, -, \*, /, \*\*, %, (, ), etc.

```
C:\Python27\python.exe
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> 8+2
10
>>> 5-7
-2
>>> 9*4
36
>>> 15/3
5
>>> 80%1000
80
>>> 2**5
32
>>>
```

3. Probar la definición de variables de varios tipos (int, float, string) y usarlas posteriormente en las operaciones del apartado anterior.

```
>>> i = 5
>>> f = 2.5
>>>
>>> s = "hola"
>>> i * f
12.5
>>> i - f
2.5
>>> i / s
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'int' and 'str'
>>>
```

4. Probar la instrucción *print* con las variables anteriores y con literales.

```
>>> print i
5
>>> print f
2.5
>>> print s
hola
>>>
```

5. Usar la coma para separar varios elementos en un mismo *print*. Probar *print "hola"; print "y adiós"*. Observar la coma detrás del primer *print*.

```
>>> print "hola", "y adiós"
hola y adiós
>>>
```

6. Probar print con cadenas, bool, int y floats.

```
>>> a = True
>>> print a
True
>>> print "cadena"
cadena
>>> print 3
3
>>> print a
True
>>>
```

7. Importar el módulo math y mostrar las funciones que incorpora

```
>>> import math
>>> help <math>
Help on built-in module math:

NAME
    math

FILE
    <built-in>

DESCRIPTION
    This module is always available. It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine <measured in radians> of x.

    acosh(...)
        acosh(x)

        Return the hyperbolic arc cosine <measured in radians> of
        x.

    asin(...)
        asin(x)

        Return the arc sine <measured in radians> of x.

    asinh(...)
        asinh(x)

        Return the hyperbolic arc sine <measured in radians> of x
```

8. Usar algunas de las funciones del módulo math.

```
>>> math.ceil(8.2)
9.0
>>> math.exp(5)
148.4131591025766
>>> math.copysign(3,8)
3.0
>>>
```

9. Intentar salir de Python con quit y con CTRL+D.

```
148.4131591025766
>>> math.copysign(3,8)
3.0
>>> quit
Use quit() or Ctrl-Z plus Return to exit
>>>
```

10. El modo interactivo es muy útil en Python y es importante desenvolverse bien en él.

## La ayuda del modo interactivo:

1. Para pedir la ayuda de una palabra clave en Python se pone entre paréntesis y la palabra clave entre comillas. Para pedir ayuda de la orden import escribimos: `help("import")`. Probar esto con `import`, `print`, etc.

```
>>> help("import")
The "import" statement
*****

import_stmt ::= "import" module ["as" name] < "," module ["as" name] >*
              | "from" relative_module "import" identifier ["as" name]
              < "," identifier ["as" name] >*
              | "from" relative_module "import" "<" identifier ["as" name]
              < "," identifier ["as" name] >* [" ," ] ">"
              | "from" module "import" "*"
module       ::= <identifier ".">* identifier
relative_module ::= "."* module | "."+
name         ::= identifier

Import statements are executed in two steps: (1) find a module, and
initialize it if necessary; (2) define a name or names in the local
namespace (of the scope where the "import" statement occurs). The
statement comes in two forms differing on whether it uses the "from"
keyword. The first form (without "from") repeats these steps for each
identifier in the list. The form with "from" performs step (1) once,
and then performs step (2) repeatedly.
```

```
>>> help("print")
The "print" statement
*****

print_stmt ::= "print" <[expression <"," expression>* [" ," ]
                  | ">>" expression [<"," expression>+ [" ," ]]>

"print" evaluates each expression in turn and writes the result in
```

```
KeyboardInterrupt
>>> help("string")
Help on module string:

NAME
    string - A collection of string operations (most are no longer used).

FILE
    c:\python27\lib\string.py

DESCRIPTION
    Warning: most of the code you see here isn't normally used nowadays.
    Beginning with Python 1.6, many of these functions are implemented as
```

2. Las instrucciones:  
`import rlcompleter, readline`  
`readline.parse_and_bind("tab:complete")`  
permiten completar instrucciones con ayuda del tabulador. Por ejemplo, después de importar el módulo `math`, escribir `math.` y pulsar dos veces el tabulador. Comprobar que salen todas las funciones de `math`.
3. De igual forma invocar con `help` la ayuda de algunas de las funciones del módulo `math`.

```
>>> help(math.ceil)
Help on built-in function ceil in module math:

ceil(...)
    ceil(x)

    Return the ceiling of x as a float.
    This is the smallest integral value >= x.

>>>
```

```
>>> help(math.copysign)
Help on built-in function copysign in module math:

copysign(...)
    copysign(x, y)

    Return x with the sign of y.
```

4. Entrar en el sistema de ayuda interactiva con help(). Ahora no es necesario escribir help() cada vez que se solicita ayuda, se pone la palabra clave o el t3pico directamente y sale la ayuda.

```
>>> help()

Welcome to Python 2.7! This is the online help utility.

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/2.7/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules",
"keywords", or "topics". Each module also comes with a one-line summary
of what it does; to list the modules whose summaries contain a given word
such as "spam", type "modules spam".

help> ceil
no Python documentation found for 'ceil'

help> math.ceil
Help on built-in function ceil in math:

math.ceil = ceil(...)
    ceil(x)

    Return the ceiling of x as a float.
    This is the smallest integral value >= x.

help>
```

5. Para ver una lista con los temas de ayuda ejecutar la orden topics.

```
help> topics

Here is a list of available topics. Enter any topic name to get more help.

ASSERTION          DEBUGGING          LITERALS          SEQUENCEMETHODS2
ASSIGNMENT         DELETION          LOOPING           SEQUENCES
ATTRIBUTEMETHODS DICTIONARIES     MAPPINGMETHODS   SHIFTING
ATTRIBUTES         DICTIOARYLITERALS MAPPINGS         SLICINGS
AUGMENTEDASSIGNMENT DYNAMICFEATURES  METHODS          SPECIALATTRIBUTES
BACKQUOTES         ELLIPSIS         MODULES          SPECIALIDENTIFIERS
BASICMETHODS       EXCEPTIONS       NAMESPACES       SPECIALMETHODS
BINARY            EXECUTION       NONE            STRINGMETHODS
BITWISE           EXPRESSIONS     NUMBERMETHODS   STRINGS
BOOLEAN          FILES          NUMBERS         SUBSCRIPTS
CALLABLEMETHODS   FLOAT         OBJECTS        TRACEBACKS
CALLS            FORMATTING     OPERATORS       TRUTHVALUE
CLASSES          FRAMEOBJECTS   PACKAGES        TUPLELITERALS
CODEOBJECTS      FRAMES        POWER           TUPLES
COERCIONS        FUNCTIONS     PRECEDENCE      TYPEOBJECTS
COMPARISON       IDENTIFIERS    PRINTING        TYPES
COMPLEX          IMPORTING     PRIVATENAMES    UNARY
CONDITIONAL      INTEGER       RETURNING       UNICODE
CONTEXTMANAGERS  LISTLITERALS  SCOPING
CONVERSIONS      LISTS        SEQUENCEMETHODS1
```

6. Solicitar ayuda de alguno de los topics anteriores.

```

help> ASSERTION
The "assert" statement
*****

Assert statements are a convenient way to insert debugging assertions
into a program:

    assert_stmt ::= "assert" expression ["," expression]

The simple form, "assert expression", is equivalent to

    if __debug__:
        if not expression: raise AssertionError

The extended form, "assert expression1, expression2", is equivalent to

    if __debug__:
        if not expression1: raise AssertionError(expression2)

These equivalences assume that "__debug__" and "AssertionError" refer
to the built-in variables with those names. In the current
implementation, the built-in variable "__debug__" is "True" under
normal circumstances, "False" when optimization is requested (command
line option -O). The current code generator emits no code for an
assert statement when optimization is requested at compile time. Note
that it is unnecessary to include the source code for the expression
that failed in the error message; it will be displayed as part of the
stack trace.

Assignments to "__debug__" are illegal. The value for the built-in
variable is determined when the interpreter starts.

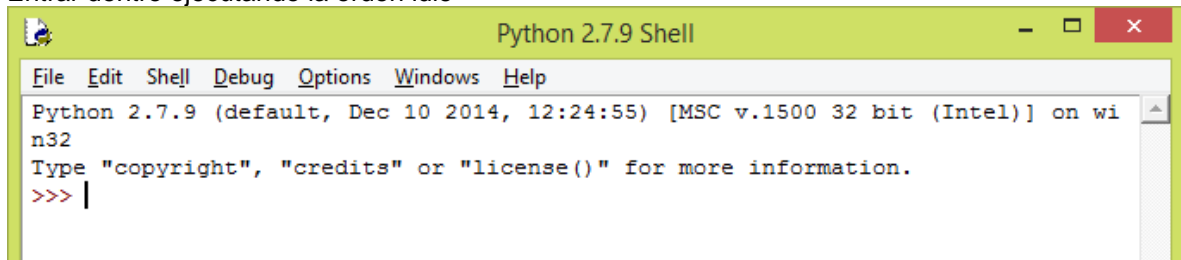
help>

```

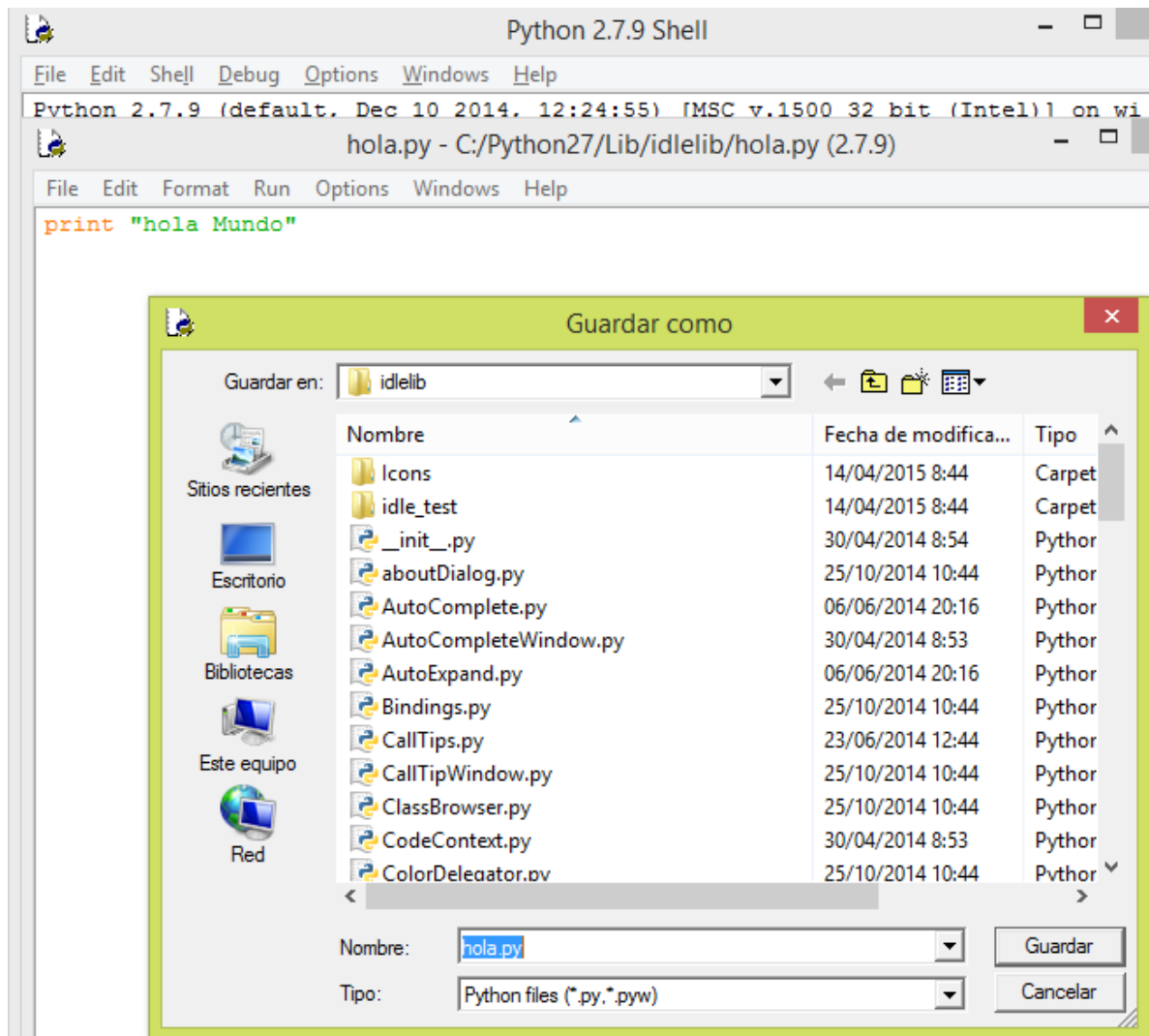
7. El sistema de ayuda es muy importante en Python, y es muy importante y conveniente familiarizarse bien con él. Nos sacará de muchos aprietos.

## El editor idle

1. Entrar dentro ejecutando la orden idle



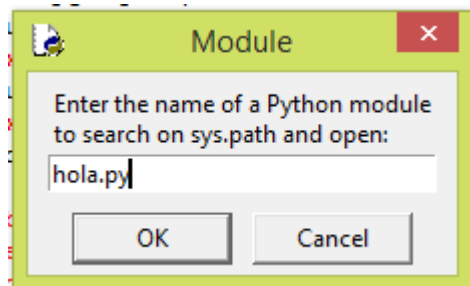
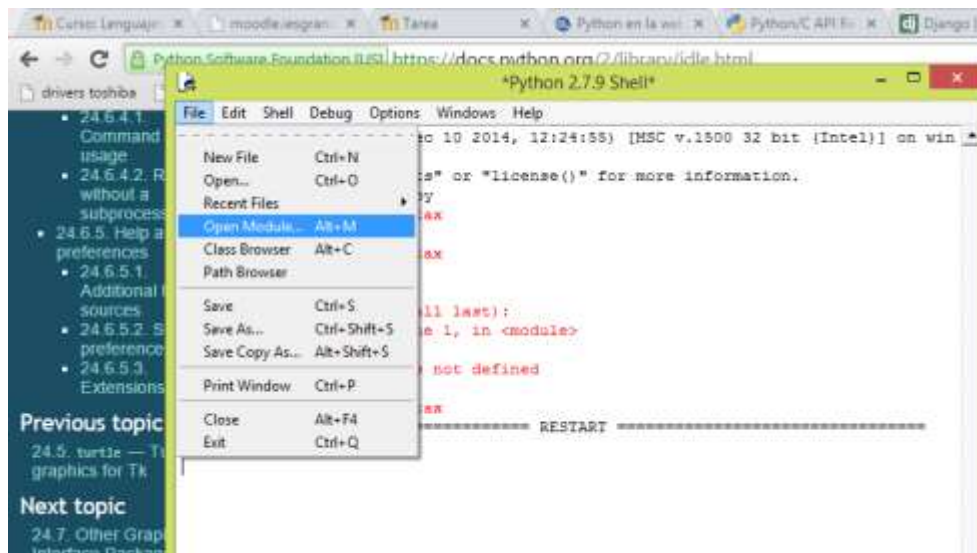
2. Crear el fichero hola.py con el típico programa que visualice la cadena "hola Mundo".



3. Guardar el fichero y ejecutar nuestro primer programa en python desde la shell de UNIX. Ojo con los permisos de ejecución y la ruta hasta python.

```
C:\Python27\Lib\idlelib>hola.py
hola Mundo
C:\Python27\Lib\idlelib>
```

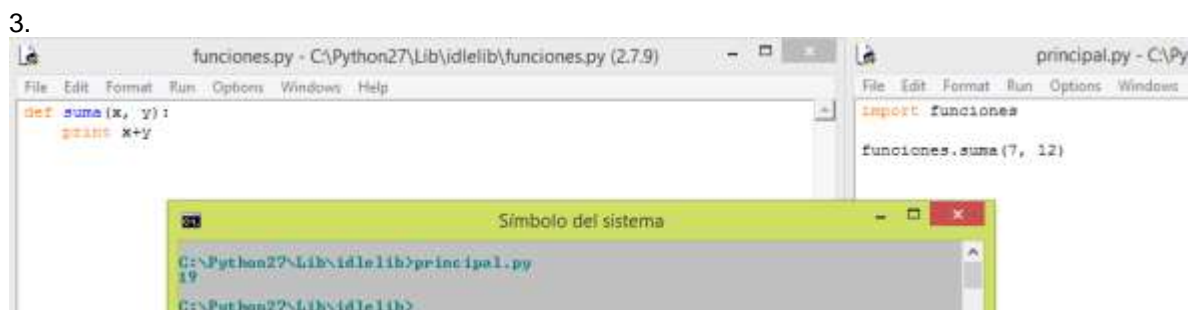
4. Ejecutar el programa anterior desde dentro de IDLE con el menú 'Run', opción 'Run Module' o bien F5.



5. Navegar los menús de IDLE y comprobar su utilidad. Este editor puede usarse con Python, pero también cualquier otro editor de código; hacer la prueba con cualquier otro editor (nedit, gedit, etc).

## Funciones

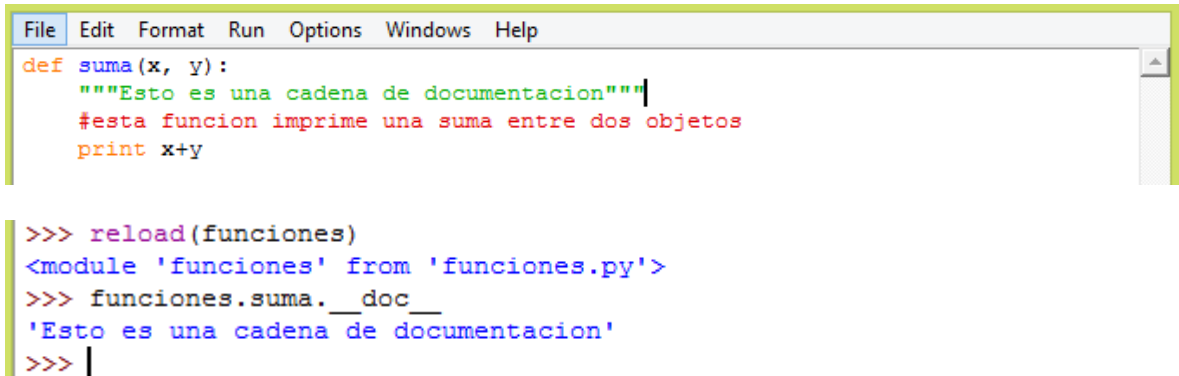
1. Crear un fichero funciones.py que contenga una función que se llame suma y que devuelva la suma de los dos parámetros y un programa principal que llame a la función anterior con dos valores cualesquiera.
2. Ejecutar el programa desde la *shell* del SO.



4. Entrar en el modo interactivo e importar el modulo funciones, y sin salir del modo interactivo invocar a la función suma con enteros, reales y cadenas . Observar que después de importar el módulo funciones, para invocar a la función suma sería: funciones.suma().

```
>>> import funciones
>>> funciones.suma(8, 10)
18
>>> funciones.suma(3.5, 4.5)
8.0
>>> funciones.suma("hola", " y adios")
hola y adios
>>> |
```

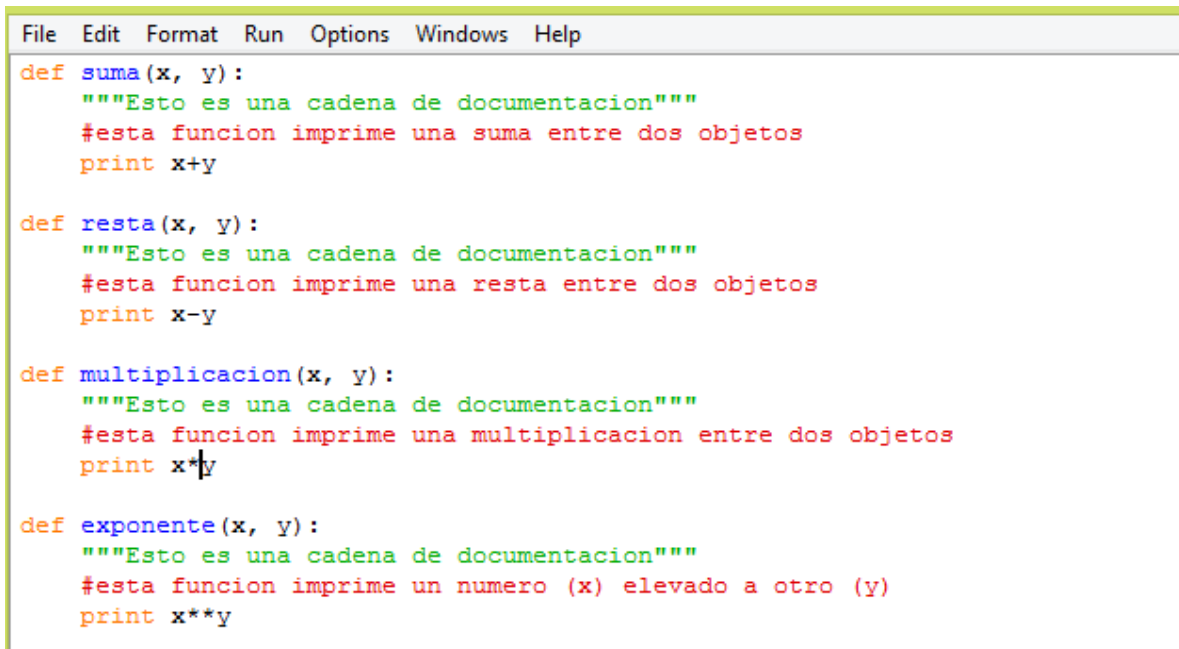
5. Añadirle una cadena de documentación a la función suma, volver a importar el módulo funciones e invocar la cadena de documentación de la función suma que acabamos de añadir. ¡Ojo!: para recargar un módulo una vez modificado usar: reload(modulo).



```
File Edit Format Run Options Windows Help
def suma(x, y):
    """Esto es una cadena de documentacion"""
    #esta funcion imprime una suma entre dos objetos
    print x+y

>>> reload(funciones)
<module 'funciones' from 'funciones.py'>
>>> funciones.suma.__doc__
'Esto es una cadena de documentacion'
>>> |
```

6. Hacer otras funciones (resta, multiplicación, etc.) probando y repitiendo lo anterior.



```
File Edit Format Run Options Windows Help
def suma(x, y):
    """Esto es una cadena de documentacion"""
    #esta funcion imprime una suma entre dos objetos
    print x+y

def resta(x, y):
    """Esto es una cadena de documentacion"""
    #esta funcion imprime una resta entre dos objetos
    print x-y

def multiplicacion(x, y):
    """Esto es una cadena de documentacion"""
    #esta funcion imprime una multiplicacion entre dos objetos
    print x*y

def exponente(x, y):
    """Esto es una cadena de documentacion"""
    #esta funcion imprime un numero (x) elevado a otro (y)
    print x**y
```



```

>>> reload(funciones)
<module 'funciones' from 'funciones.py'>
>>> funciones.resta(8, 2)
6
>>> funciones.multiplicacion(5, 8)
40
>>> funciones.exponente(2, 6)
64
>>> |

```

## Cadenas

1. En el modo interactivo, crear un par de variables con sendas cadenas y construir una cadena nueva a partir las otras dos anteriores mediante concatenación.

```

>>> cadena1 = "hola"
>>> cadena2 = " y adios"
>>> cadena3 = cadena1 + cadena2
>>> print cadena3
hola y adios
>>> |

```

2. Usar el *slicing* de cadenas para obtener: el principio de una cadena hasta cierta posición, el final de una cadena desde cierta posición, una subcadena desde una posición a otra, etc.

```

>>> cadena3[0:4]
'hola'
>>> cadena3[-5:]
'adios'
>>> cadena3.find("y adi")
5
>>> |

```

3. Intentar acceder a una posición no existente de una cadena.

```

>>> cadena3[12]

Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    cadena3[12]
IndexError: string index out of range
>>> |

```

4. Modificar el valor de una variable de tipo cadena previamente creada.

```
>>> cadena = "hola"
>>> cadena
'hola'
>>> cadena = "adios"
>>> cadena
'adios'
>>> |
```

5. Comprobar la utilidad de upper, lower, strip, max (con una y varias cadenas), min (con una y varias cadenas), n, not in, etc.

```
>>> cadena = "hola"
>>> cadena
'hola'
>>> cadena.upper()
'HOLA'
>>> cadena.lower()
'hola'
>>> cadena2 = " adios  "
>>> cadena2
' adios '
>>> cadena2.strip()
'adios'
>>> max(cadena)
'o'
>>> min(cadena)
'a'
>>> max(cadena, cadena2)
'hola'
>>> cadena3 = "hasta pronto"
>>> min(cadena, cadena2, cadena3)
' adios '
>>> |
```

```
>>> cadena1 = "hola"
>>> cadena1 = "hola y adios"
>>> cadena2 = "adios"
>>> cadena2 in cadena1
True
>>> cadena1 not in cadena2
True
>>> cadena1 in cadena2
False
>>> |
```

6. Importar el módulo string. Usar: letters, lowercase, uppercase, digits, punctuation, y el resto de utilidades. Acudir a la ayuda para aprender a usarlas.

```

--
>>> print string.digits
0123456789
>>> cadena1 = "hola, adios"
>>> indice = string.find(cadena1, ",")
>>> print indice
4
>>> string.find("Hola", "a", 1, 2)
-1
>>> find(lowercase, carac)

Traceback (most recent call last):
  File "<pyshell#130>", line 1, in <module>
    find(lowercase, carac)
NameError: name 'find' is not defined
>>> print string.lowercase
abcdefghijklmnopqrstuvwxyzšœž*µ°ßàáâãäåæçèéêëìíîïðñòóôõöøùúûüýþ
>>> print string.uppercase
ABCDEFGHIJKLMNOPQRSTUVWXYZŠŒŽYÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝÞ
>>> print string.punctuation
!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~

```

7. Importar el módulo string. Ver la ayuda de la función string.split(). mediante la orden help(string.split) y probarla en el modo interactivo. Hacer lo mismo con find, replace, join.

```

...
>>> help(string.split)
Help on function split in module string:

split(s, sep=None, maxsplit=-1)
    split(s [,sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string s, using sep as the
    delimiter string.  If maxsplit is given, splits at no more than
    maxsplit places (resulting in at most maxsplit+1 words).  If sep
    is not specified or is None, any whitespace string is a separator.

    (split and splitfields are synonymous)

>>> help(string.find)
Help on function find in module string:

find(s, *args)
    find(s, sub [,start [,end]]) -> in

    Return the lowest index in s where substring sub is found,
    such that sub is contained within s[start,end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.

>>> help(string.replace)
Help on function replace in module string:

replace(s, old, new, maxreplace=-1)
    replace (str, old, new[, maxreplace]) -> string

    Return a copy of string str with all occurrences of substring
    old replaced by new.  If the optional argument maxreplace is
    given, only the first maxreplace occurrences are replaced.

>>> help(string.join)
Help on function join in module string:

join(words, sep=' ')
    join(list [,sep]) -> string

```

---

```

>>> string.split(cadena1)
['hola,', 'adios']
>>> cadena1.find("a")
3
>>> cadena1.join(cadena2)
'ahola, adiosdhola, adiosihola, adiosohola, adioss'
>>> cadena1
'hola, adios'
>>> cadena1.replace("h", "L")
'Lola, adios'
>>>

```

8. Pedir al usuario una frase y mostrar en pantalla el número de palabras de esa frase.

```
File Edit Format Run Options Windows Help
cadena = raw_input('Escribe una frase: ')
while cadena != 'q':
    cambios = 0
    for i in range(1, len(cadena)):
        if cadena[i] == ' ' and cadena[i-1] != ' ':
            cambios = cambios + 1

    if cadena[-1] == ' ':
        cambios = cambios - 1

    palabras = cambios + 1
    print 'Palabras:', palabras

    print ""q"", "para salir"

    cadena = raw_input('Escribe una frase: ')
```

```
C:\Python27\Lib\idlelib>pidfrase.py
Escribe una frase: hola y adios
Palabras: 3
q para salir
Escribe una frase: q
C:\Python27\Lib\idlelib>
```