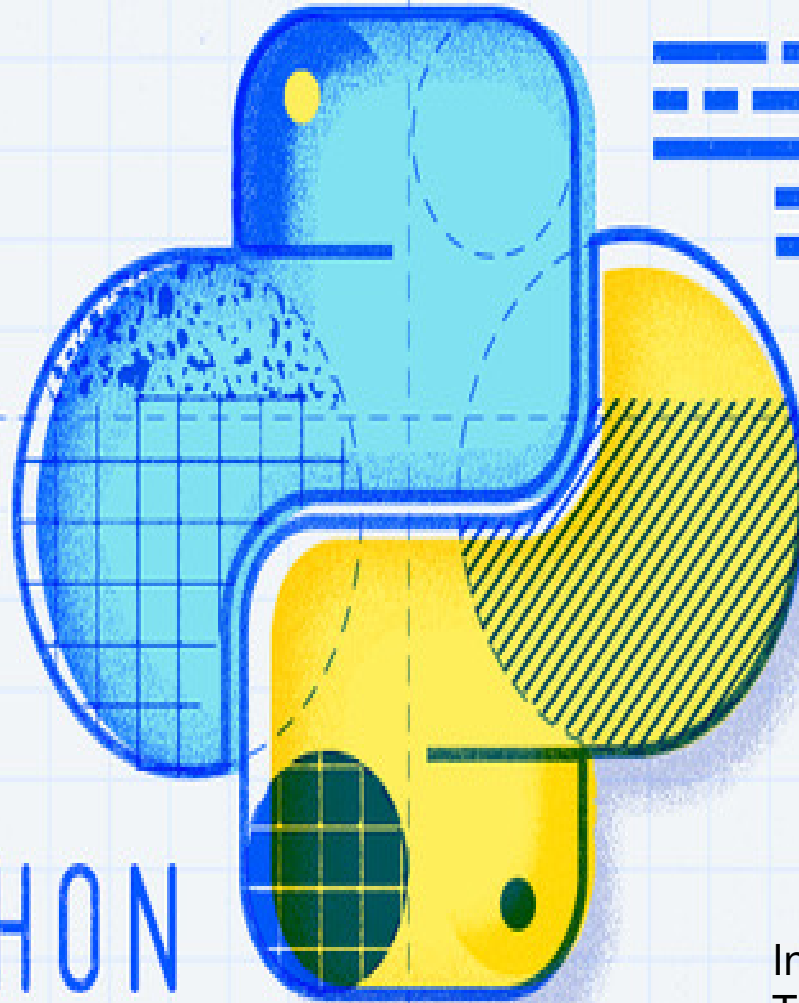




PYTHON



Ing. Leonardo Renteria PhD.
Tel: 0960450234
email: lfreneriax@hotmail.com

@lfreneriax



**Requisitos:**

- Conocimientos básicos en programación (de preferencia).
- Computador
- Conexión a internet



Metodología

Actividad	Horas Presenciales	Horas no presenciales
A-1 Clases expositivas/participativas	16	
A-2 Prácticas	16	
A-3 Estudio y trabajo autónomo del estudiante		4
A-4 Pruebas de evaluación	4	
Total	36	4



Contenidos:



Tema	Nro. Horas	Día
1. Introducción al lenguaje de programación	1	D1
2. Instalación de Python	0,5	
3. Modo interactivo	1	
4. Sintaxis básica	1.5	
5. Tipos de variables	1	D2
6. Números	1	
7. Strings	2	



Contenidos:



Tema	Nro. Horas	Día
8. Tuplas	1	D3
9. Diccionarios	1	
10 Listas	2	
11. Operadores básicos	3	D4
12. Scripts	1	
13. Toma de decisiones	3	D5
14. Control de errores	1	



Contenidos:



Tema	Nro. Horas	Día
15. Bucles	4	D6
16. Funciones	2	D7
17. Módulos	2	
18. Archivos	4	D8
19. Estudio y trabajo autónomo del estudiante	4	D9
20. Prueba de evaluación	4	D10





lenguajes de programación?

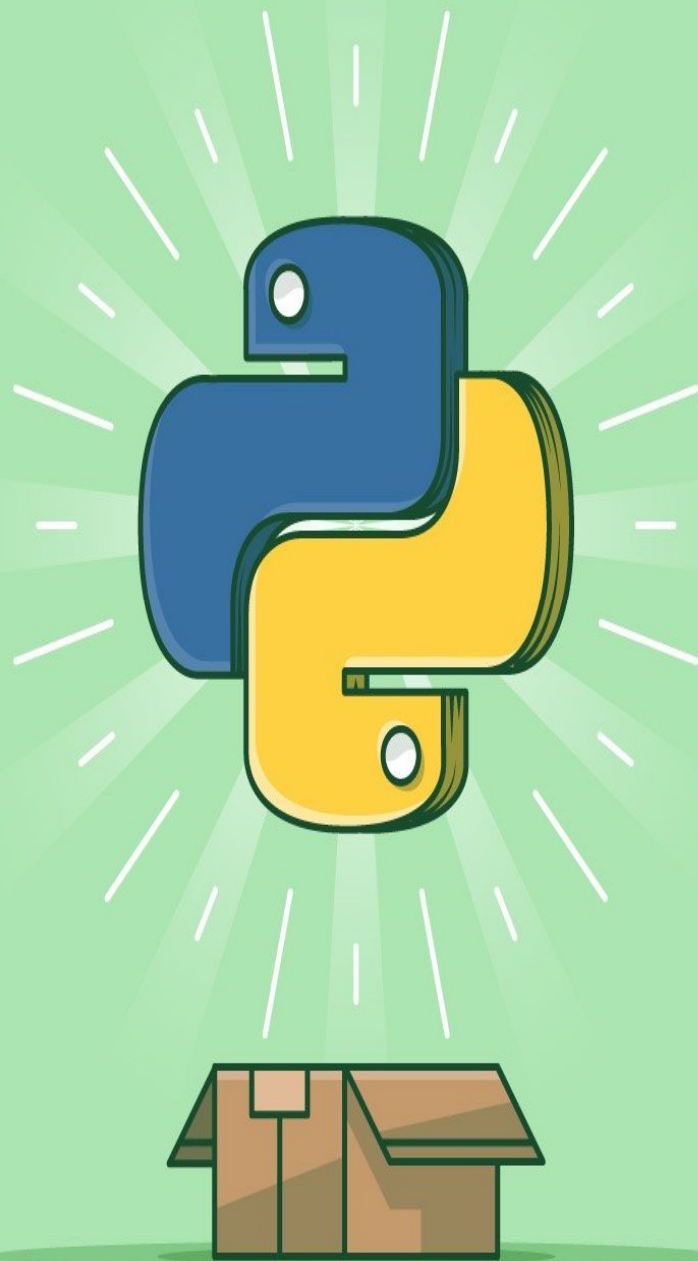




python™

lenguajes de programación?







Python es un lenguaje de programación de propósito general muy poderoso y flexible, a la vez que sencillo y fácil de aprender

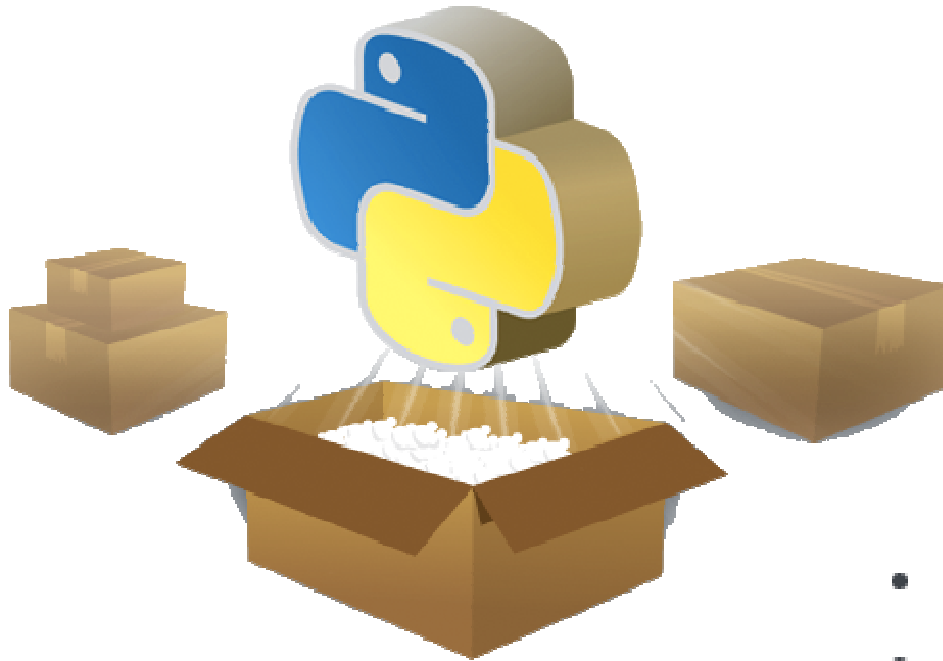




Python es un lenguaje de programación de propósito general muy poderoso y flexible, a la vez que sencillo y fácil de aprender



Python Package Index (PyPI)



- Desarrollo web e Internet.
- Acceso a la base de datos.
- GUIs de escritorio.
- Científico y numérico.
- Educación.
- Programación de red.
- Desarrollo de Software y Juegos.



Características

- Es un lenguaje interpretado, **no compilado**, usa tipado dinámico, fuertemente tipado.
- Es multiplataforma, lo cual es ventajoso para hacer ejecutable su código fuente entre varios sistema operativos.
- Es un lenguaje de programación multiparadigma, el cual soporta varios paradigma de programación como orientación a objetos, estructurada, programación imperativa y, en menor medida, programación funcional.
- En Python, el formato del código (p. ej., la indentación) es estructural.



Fuertemente tipado

El fuertemente tipado significa que el tipo de valor no cambia repentinamente. Un string que contiene solo dígitos no se convierte mágicamente en un número. Cada cambio de tipo requiere una conversión explícita. A continuación un ejemplo de este concepto:

```
# variable "valor1" es integer, variable "valor2" es string  
valor1, valor2 = 2, "5"  
# el metodo int() es para convertir a integer  
total = valor1 + int(valor2)  
# el metodo str() es para convertir a string  
print "El total es: " + str(total)
```



Tipado dinámico

El tipado dinámico significa que los objetos en tiempo de ejecución (valores) tienen un tipo, a diferencia del tipado estático donde las variables tienen un tipo. A continuación un ejemplo de este concepto:

```
# "variable" guarda un valor integer
variable = 11
print variable, type(variable)
# "variable" guarda un valor string
variable = "activo"
print (variable), type(variable)
```



Filosofía “Incluye baterías”

- Python ha mantenido durante mucho tiempo esta filosofía de “baterías incluidas”:

“Tener una biblioteca estándar rica y versátil que está disponible de inmediato. Sin que el usuario descargue paquetes separados.”

- Esto le da al lenguaje una ventaja en muchos proyectos.

Zen de Python”

```
>>> import this
```



Es una colección de 20 principios de software que influyen en el diseño del Lenguaje de Programación Python, de los cuales 19 fueron escritos por *Tim Peters* en junio de 1999. El texto es distribuido como dominio público.

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!



Ventajas

Simplificado y rápido

Este lenguaje simplifica mucho la programación “hace que te adaptes a un modo de lenguaje de programación, Python te propone un patrón”. Es un gran lenguaje para scripting, si usted requiere algo rápido (en el sentido de la ejecución del lenguaje), con unas cuantas líneas ya está resuelto.

Elegante y flexible

El lenguaje le da muchas herramientas, si usted quiere listas de varios tipos de datos, no hace falta que declares cada tipo de datos. Es un lenguaje tan flexible que usted no se preocupa tanto por los detalles.

Programación sana y productiva

Programar en Python se convierte en un estilo muy sano de programar: es sencillo de aprender, direccionado a las reglas perfectas, le hace como dependiente de mejorar, cumplir las reglas, el uso de las líneas, de variables”. Además es un lenguaje que fue hecho con productividad en mente, es decir, Python le hace ser más productivo, le permite entregar en los tiempos que se requieren.



Ventajas

Portable

Es un lenguaje muy portable (ya sea en Mac, Linux o Windows) en comparación con otros lenguajes. La filosofía de baterías incluidas, son las librerías que más usted necesita al día a día de programación, ya están dentro del interprete, no tiene la necesidad de instalarlas adicionalmente con en otros lenguajes.

Comunidad

Algo muy importante para el desarrollo de un lenguaje es la comunidad, la misma comunidad de Python cuida el lenguaje y casi todas las actualizaciones se hacen de manera democrática.



Desventajas

Curva de aprendizaje

La “curva de aprendizaje cuando ya estás en la parte web no es tan sencilla”.

Hosting

La mayoría de los servidores no tienen soporte a Python, y si lo soportan, la configuración es un poco difícil.

Librerías incluidas

Algunas librerías que trae por defecto no son del gusto de amplio de la comunidad, y optan a usar librerías de terceros.



Instalación

https://tutorial.djangogirls.org/es/python_installation/



shorturl.at/qtJOV



```
>>> 1 + 2
3
>>> name = "Sarah"
>>> "Hello " + name
'Hello Sarah'
```

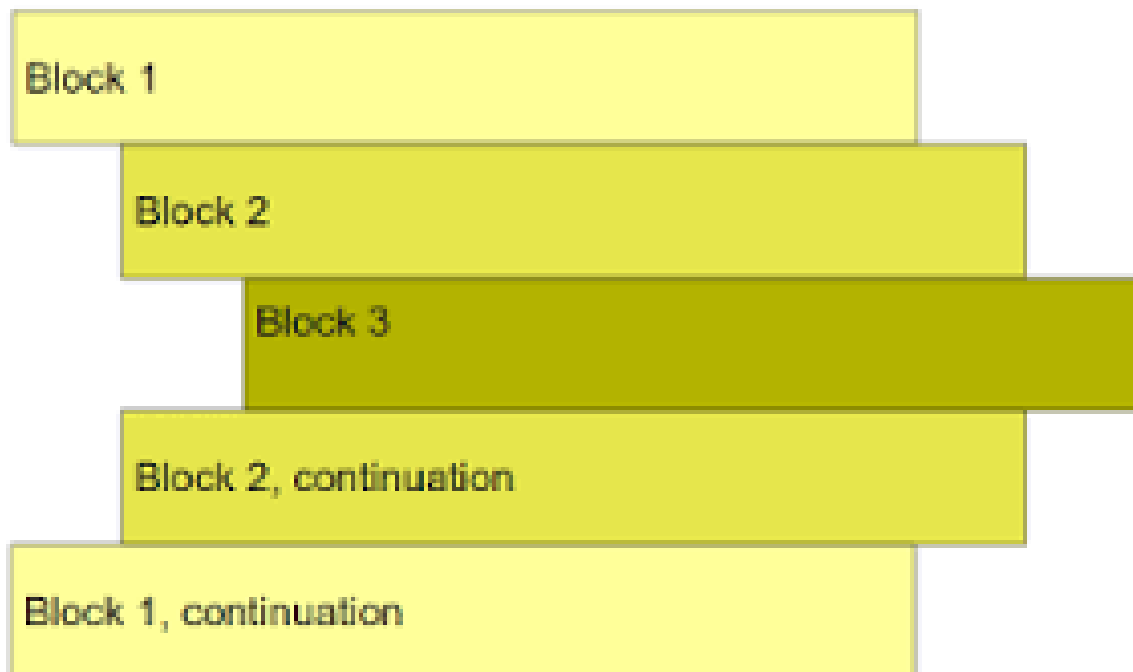
```
print("Hello world")
```



```
1 #include <stdio.h>
2
3 #define SIZE 4
4
5 int main()
6 {
7     char *strings[SIZE] =
8         {
9             "String1",
10            "String2",
11            "String3",
12            "String4"
13        };
14
15     char *ptr_swap; /* A temporary pointer to swap strings */
16
17     /* Swap "String2" with 'String3' */
18     ptr_swap = strings [1];
19     strings [1] = strings [2];
20     strings [2] = ptr_swap;
21
22     printf ("%s %s %s %s", strings[0], strings[1], strings[2], strings[3]);
23
24     return 0;
25 }
```

```
def f(n):
    if n == 1:
        return 1
    else:
        return f(n-1)

print(f(4))
```



```
def f(n):  
    if n == 1:  
        return 1  
    else:  
        return f(n-1)  
  
print(f(4))
```



python™

```
process_string.py x
1 import csv
2 import re
3
4 with open("search_file.csv") as source, open("list.csv") as module_names, open("Final_File.csv",
5     reader=csv.reader(source)
6     module=csv.reader(module_names)
7     writer=csv.writer(result)
8     flag=False
9     for row in reader:
10         i=row[1]
11         for s in module_names:
12             k=s[0].strip()
13             l=s[1].strip()
14             print(k)
15             if i.find(k) != -1 and flag==False:
16                 row[1]=l
17                 writer.writerow(row)
18                 flag=True
19             module_names.seek(0)
20             flag=False
21
```

```
def f(n):
    if n == 1:
        return 1
    else:
        return f(n-1)

print(f(4))
```



```
process_string.py x
1 import csv
2 import re
3
4 with open("search_file.csv") as source:
5     reader = csv.reader(source)
6     module = csv.reader(module_names)
7     writer = csv.writer(result)
8     flag = False
9     for row in reader:
10         i = row[1]
11         for s in module_names:
12             k = s[0].strip()
13             l = s[1].strip()
14             print(k)
15             if i.find(k) != -1 and flag
```

```
def f(n):
    if n == 1:
        return 1
    else:
        return f(n-1)

print(f(4))
```



```
for i in range(10):  
    print("Hello")
```

```
for i in range(2):  
    print("A")  
    print("B")
```



```
name = "Jamie"  
print(len(name)) # 5
```

```
names = ["Bob", "Jane", "James", "Alice"]  
print(len(names)) # 4
```

```
name = "Joe"  
  
if len(name) > 3:  
    print("Nice name,")  
    print(name)  
else:  
    print("That's a short name,")  
    print(name)
```



Modo interactivo

La *inmersión al modo interactivo* le permite a cualquier usuario el cual **NUNCA** ha trabajado con el interprete de Python pueda tener un primer acercamiento **SIN PROGRAMAR**, solamente con conocer el uso del interprete y sus comandos básicos usando la técnica de introspección.

```
python
Python 2.7.13 (default, Sep 26 2018, 18:42:22)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



Modo interactivo

>>>help

```
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> 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".



Modo interactivo

help>modules

```
help> modules
```

```
Please wait a moment while I gather a list of all available modules...
```

BaseHTTPServer	asynchat	imputil	sha
Bastion	asyncore	inspect	shelve
CDROM	atexit	io	shlex
CGIHTTPServer	audiodev	ipython_genutils	shutil
Canvas	audioop	itertools	shutil_backport
ConfigParser	autoreload	jinja2	signal
Cookie	babel	json	simplegeneric
DLFCN	backports	keyword	site
Dialog	base64	lib2to3	sitecustomize
DocXMLRPCServer	bdb	linecache	six
FileDialog	binascii	linuxaudiodev	smtpd
FixTk	binhex	locale	smtplib
HTMLParser	bisect	logging	sndhdr
IN	bsddb	macpath	snowballstemmer
IPython	bz2	macurl2path	socket
MimeWriter	cPickle	mailbox	sphinx



Modo interactivo

help>os

```
help> os
Help on module os:

NAME
    os - OS routines for NT or Posix depending on what system we're on.

FILE
    /usr/lib/python2.7/os.py

MODULE DOCS
    https://docs.python.org/library/os

DESCRIPTION
    This exports:
      - all functions from posix, nt, os2, or ce, e.g. unlink, stat, etc.
      - os.path is one of the modules posixpath, or ntpath
      - os.name is 'posix', 'nt', 'os2', 'ce' or 'riscos'
      - os.curdir is a string representing the current directory ('.' or ':')
      - os.pardir is a string representing the parent directory ('..' or '..:')
      - os.sep is the (or a most common) pathname separator ('/' or ':' or '\')
      - os.extsep is the extension separator ('.' or '/')
      - os.altsep is the alternate pathname separator (None or '/')
```



Modo interactivo

help>os

```
help> os
Help on module os:

NAME
    os - OS routines for NT or Posix depending on what system we're on.

FILE
    /usr/lib/python2.7/os.py

MODULE DOCS
```

Truco:

Presione la tecla q para salir de la ayuda del módulo os.



Modo interactivo

>>>import os

```
>>> import os
>>>
```

>>>dir(os)

```
>>> dir(os)
['EX_CANTCREAT', 'EX_CONFIG', 'EX_DATAERR', 'EX_IOERR', 'EX_NOHOST',
'EX_NOINPUT', 'EX_NOPERM', 'EX_NOUSER', 'EX_OK', 'EX_OSERR', 'EX_OSFILE',
'EX_PROTOCOL', 'EX_SOFTWARE', 'EX_TEMPFAIL', 'EX_UNAVAILABLE',
'EX_USAGE', 'F_OK', 'NGROUPS_MAX', 'O_APPEND', 'O_CREAT', 'O_DIRECT',
'O_DIRECTORY', 'O_DSYNC', 'O_EXCL', 'O_LARGEFILE', 'O_NDELAY',
'O_NOCTTY', 'O_NOFOLLOW', 'O_NONBLOCK', 'O_RDONLY', 'O_RDWR', 'O_RSYNC',
'O_SYNC', 'O_TRUNC', 'O_WRONLY', 'P_NOWAIT', 'P_NOWAITO', 'P_WAIT',
'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET', 'TMP_MAX', 'UserDict',
'WCONTINUED', 'WCOREDUMP', 'WEXITSTATUS', 'WIFCONTINUED', 'WIFEXITED',
'WIFSIGNALED', 'WIFSTOPPED', 'WNOHANG', 'WSTOPSIG', 'WTERMSIG',
'WUNTRACED', 'W_OK', 'X_OK', '_Environ', '__all__', '__builtins__',
'__doc__', '__file__', '__name__', '_copy_reg', '_execvpe', '_exists',
'_exit', '_get_exports_list', '_make_stat_result',
```



Modo interactivo

```
>>>os.__file__
```

```
>>> os.__file__  
'/usr/lib/python2.7/os.pyc'  
>>>
```

```
>>>print (os.__doc__)
```

```
>>> print os.__doc__  
OS routines for NT or Posix depending on what system we're on.  
  
This exports:  
- all functions from posix, nt, os2, or ce, e.g. unlink, stat, etc.  
- os.path is one of the modules posixpath, or ntpath  
- os.name is 'posix', 'nt', 'os2', 'ce' or 'riscos'  
- os.curdir is a string representing the current directory ('.' or ':')  
- os.pardir is a string representing the parent directory ('..' or '::')  
- os.sep is the (or a most common) pathname separator ('/' or ':' or '\\')  
- os.extsep is the extension separator ('.' or '/')  
- os.pathsep is the path separator (usually ':')  
- os.defpath is the default search path (usually '')  
- os.linesep is the line separator (usually '\n')  
- os.altsep is an alternative pathname separator (usually None)
```



Modo interactivo

```
>>>os.__file__
```

```
>>> os.__file__  
'/usr/lib/python2.7/os.pyc'  
>>>
```

```
>>>print (os.__doc__)
```

```
>>> print os.__doc__  
OS routines for NT or Posix depending on what system we're on.  
  
This exports:  
- all functions from posix, nt, os2, or ce, e.g. unlink, stat, etc.  
- os.path is one of the modules posixpath, or ntpath  
- os.name is 'posix', 'nt', 'os2', 'ce' or 'riscos'  
- os.curdir is a string representing the current directory ('.' or ':')  
- os.pardir is a string representing the parent directory ('..' or '::')  
- os.sep is the (or a most common) pathname separator ('/' or ':' or '\\')  
- os.extsep is the extension separator ('.' or '/')  
- os.altsep is the alternate pathname separator (None or '/')  
- os.pathsep is the path separator (None or the first of several (';', ' ', or other))  
- os.usrpath is the path to the user's home directory (None or path)
```



Modo interactivo

Interprete ipython

“ipython es un shell interactivo que añade funcionalidades extra al modo interactivo incluido con Python, como resaltado de líneas y errores mediante colores, una sintaxis adicional para el shell, completado automático mediante tabulador de variables, módulos y atributos; entre otras funcionalidades. Es un componente del paquete SciPy.”

```
sudo apt-get install ipython
```



Modo interactivo

Interprete ipython

```
ipython
Python 2.7.13 (default, Sep 26 2018, 18:42:22)
Type "copyright", "credits" or "license" for more information.

IPython 5.8.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]:
```



Modo interactivo

Interprete ipython

```
In [1]: help(dir)
Help on built-in function dir in module __builtin__:

dir(...)
    dir([object]) -> list of strings

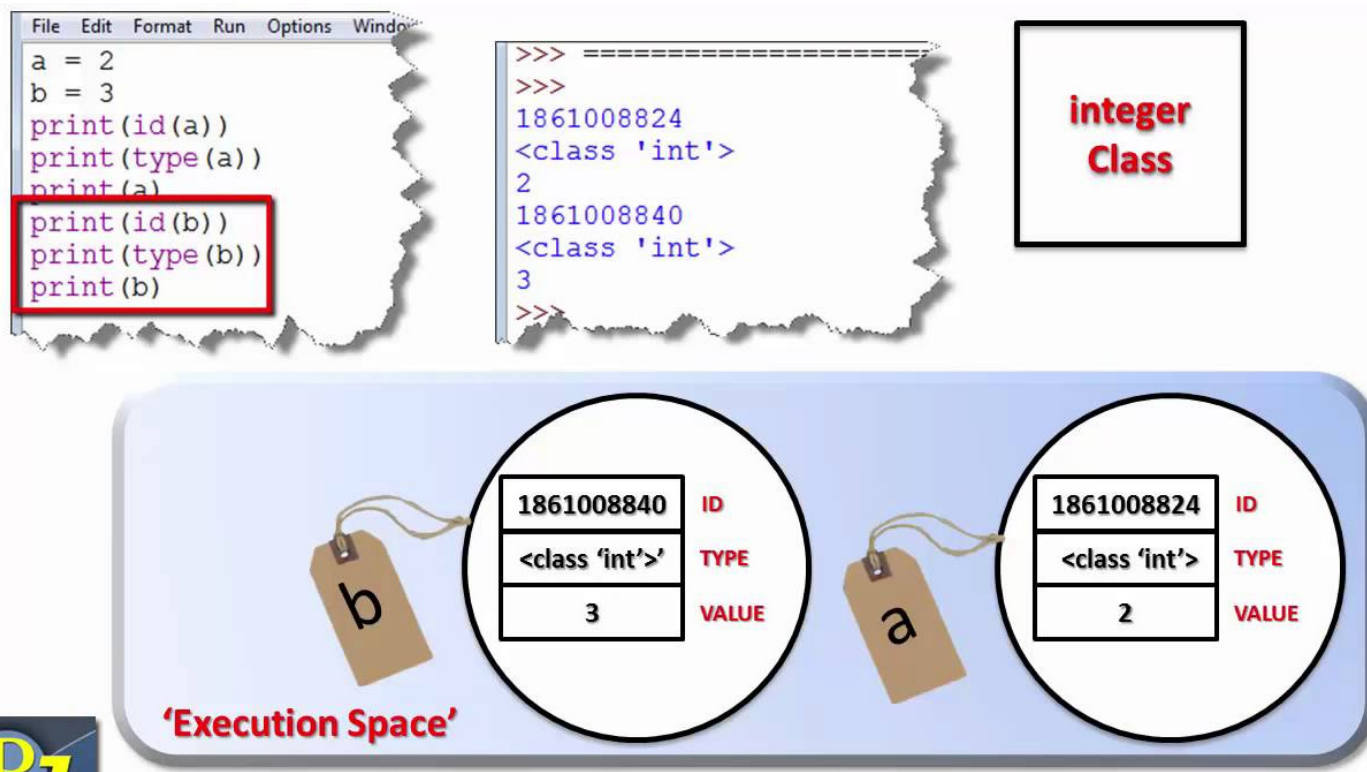
    Return an alphabetized list of names comprising (some of) the
    attributes of the given object, and of attributes reachable
    from it:

    No argument: the names in the current scope.
    Module object: the module attributes.
    Type or class object: its attributes, and recursively the
    attributes of its bases.
    Otherwise: its attributes, its class's attributes, and
    recursively the attributes of its class's base classes.
```

Tipos de datos

En Python tiene varios tipos de datos *compuestos* estándar disponibles por defecto en el interprete, como los tipos *numéricos*, *secuencias*, *mapeos* y *conjuntos* usados para agrupar otros valores.

Para el caso de las estructuras de datos se usan variables y constantes las cuales usan operadores para tratar los tipos de datos estándar.





Tipos de datos

Clasificación

Mutable

Los tipos de datos mutables son todos aquellos a los que es posible cambiar, modificar o actualizar su contenido. Los objetos mutables más comunes son las listas, diccionarios y conjuntos que sirven para guardar colecciones de datos.

No Mutable

No es posible cambiar, modificar o actualizar el contenido a un tipo de dato inmutable, aunque como en cualquier variable sí es posible asignarle un nuevo valor.



Tipos de datos

Clasificación

Categoría de tipo	Nombre	Descripción
Números inmutables	int	<u>entero</u>
	long	<u>entero long</u>
	float	<u>coma flotante</u>
	complex	<u>complejo</u>
	bool	<u>booleano</u>
Secuencias inmutables	str	<u>cadena de caracteres</u>
	unicode	<u>cadena de caracteres Unicode</u>
	tuple	<u>tupla</u>
	xrange	<u>rango inmutable</u>
Secuencias mutables	list	<u>lista</u>
	range	<u>rango mutable</u>
Mapeos	dict	<u>diccionario</u>
Conjuntos mutables	set	<u>conjunto mutable</u>
Conjuntos inmutables	frozenset	<u>conjunto inmutable</u>



Tipos de datos

Clasificación

Categoría de tipo	Nombre	Descripción
Objeto integrado	NoneType	el objeto <u>None</u> .
Objeto integrado	NotImplementedType	el objeto <u>NotImplemented</u> .
Objeto integrado	ellipsis	el objeto <u>Ellipsis</u> .
Objeto integrado	file	el objeto <u>file</u> .



Tipos de datos

Immutable

```
bpython
>>> cadena = 'Hola mundo'
>>> print cadena
Hola mundo
>>>
```

Si se quiere cambiar la letra m y sustituirla por M, no es posible con este tipo de dato, lo que se puede hacer es asignar la nueva cadena a esa misma variable.

```
bpython
>>> cadena = 'Hola mundo'
>>> print cadena
Hola mundo
>>> cadena[6] = 'M'
```



Tipos de datos

Immutable

```
bpython
>>> cadena = 'Hola mundo'
>>> print cadena
Hola mundo
>>>
```

Si se quiere cambiar la letra m y sustituirla por M, no es posible con este tipo de dato, lo que se puede hacer es asignar la nueva cadena a esa misma variable.

1. `cadena = 'Hola Mundo'`
2. `print cadena`



Tipos de datos

Immutable

```
1. numero = 10
2. print numero
3. type(numero)
```

```
1. entero_largo = 10L
2. print entero_largo
3. type(entero_largo)
```

```
1. real = 10.10
2. print real
3. type(real)
```



Tipos de datos

Immutable

```
1. complejo = 10.7 + 7.10j  
2. print complejo  
3. type(complejo)
```

```
1. booleano = bool(7 or 10)  
2. print booleano  
3. type(booleano)
```

```
1. tupla = (1,2,3,4,5)  
2. print tupla  
3. type(tupla)
```



Tipos de datos

Immutable

frozenset

Representa un conjunto inmutable, es un conjunto que no puede ser modificado después de haberlo creado. Hago particular énfasis en este tipo de dato ya que puede ser confundido con el tipo de dato *set* que representa un conjunto mutable, del cual les hablaré en la siguiente nota.

```
1. conjunto_estatico = frozenset({1,2,3,4,5})
2. print conjunto_estatico
3. type(conjunto_estatico)
```

```
1. conjunto_estatico = frozenset([1,2,3,4,5])
2. print conjunto_estatico
3. type(conjunto_estatico)
```

```
1. conjunto_estatico = frozenset( [1,2,3,4,5] )
2. print conjunto_estatico
3. type(conjunto_estatico)
```



Tipos de datos

Immutable

frozenset

Es importante colocar el conjunto como una colección ya sea con llaves {}, con corchetes [], con paréntesis () o entre comillas «»»; ya que este tipo de dato toma a todo el conjunto como un solo elemento y si prescindimos de estos, el intérprete pensará que le estamos pasando varios argumentos y nos lanzará un error.

```
bpython
>>> conjunto_estatico = frozenset((1,2,3,4,5))
>>> print conjunto_estatico
frozenset([1, 2, 3, 4, 5])
>>> type(conjunto_estatico)
<type 'frozenset'>
>>> conjunto_estatico = frozenset("1 2 3 4 5")
>>> print conjunto_estatico
frozenset([' ', '1', '3', '2', '5', '4'])
>>> type(conjunto_estatico)
<type 'frozenset'>
>>>
```



Tipos de datos

Mutable

```
1. lista = [dato1, dato2,..., datoN]
```

```
1. lista = ["lunes", "martes", "miércoles", "jueves", "viernes"]
```

```
1. lista = []
```

```
1. lista = list()
```



Tipos de datos

Listas

Métodos

`append(dato)` Agrega un elemento al final de la lista.

`count(dato)` Devuelve el número de ocurrencias del elemento.

`extend(lista)` Extiende la lista agregando todos los elementos de otra lista dada.

`index(dato)` Devuelve la primera posición en la que se encuentra ese dato.

`insert(posición, dato)` Inserta un elemento en la posición dada.

`pop(posición)` Quita el elemento de la posición dada y lo devuelve. Si no se indica la posición actúa sobre el último elemento.

`remove(dato)` Elimina la primera ocurrencia del dato, lanza una excepción en caso de no encontrarlo.

`reverse()` Invierte los elementos de la lista.

`sort()` Por defecto ordena de forma ascendente los elementos de la lista. Aunque también es posible indicarle en que sentido ordene los elementos.



Tipos de datos

Mutable

```
1.  numeros = [1, 2, 3, 4, 5, 6]
2.  numeros.append(7)
3.  numeros.count(3)
4.  numeros.extend([8, 9, 10])
5.  otros_numeros = [11, 12, 13]
6.  numeros.extend(otros_numeros)
7.  numeros.index(8)
8.  numeros.insert(7, 8)
9.  numeros.remove(8)
10. numeros.pop(7)
11. numeros.pop()
12. numeros.reverse()
13. numeros.sort()
14. numeros.sort(reverse=True)
```



Tipos de datos

Mutable

```
1. diccionario = {clave1:valor1, clave2:valor2, ...,claveN:valorN}
```

```
1. diccionario = {1:'enero', 2:'febrero', 3:'marzo', 4:'abril'}
```

```
1. diccionario = {}
```

```
1. diccionario = dict()
```

Tipos de datos



Diccionario

Métodos

`clear()` Elimina todos los elementos del diccionario.

`copy()` Hace una copia superficial del diccionario.

`fromkeys(claves, valor)` Crea un nuevo diccionario a partir de una colección de claves y un valor. En caso de que no se proporcione el valor por defecto pone `NONE`.

`get(clave)` Obtiene el valor de un elemento de la clave dada.

`has_key(clave)` Devuelve verdadero si el diccionario contiene la clave, en otro caso, devuelve falso.

`items()` Lista todos los elementos del diccionario en forma de tuplas.

`iteritems()` > Devuelve un iterador de los elementos del diccionario (clave, valor).

`iterkeys()` Devuelve un iterador de las llaves del diccionario.

`itervalues()` Devuelve un iterador de los valores del diccionario.

`keys()` Lista las claves del diccionario.



Tipos de datos

Diccionario

Métodos

`pop(clave)` Quita el elemento que coincide con la clave dada y devuelve el valor correspondiente.

`popitem()` Quita el primer elemento y devuelve un par (clave, valor) como una tupla.

`update()` Actualiza el diccionario a partir de claves y datos dados, puede ser a través de otro diccionario o de una lista de tuplas.

`values()` Lista los valores del diccionario.

`viewitems()` Devuelve un objeto similar a un conjunto con los elementos del diccionario.

`viewkeys()` Devuelve un objeto similar a un conjunto con las llaves del diccionario.

`viewvalues()` Devuelve un objeto similar a un conjunto con los valores del diccionario.



Tipos de datos

Diccionario

```
1. meses = {1:'enero', 2:'febrero', 3:'marzo', 4:'abril'}
2. meses.get(2)
3. meses.has_key(3)
4. meses.has_key(10)
```

```
1. meses.items()
2. meses.keys()
3. meses.values()
```

```
1. meses.pop(3)
2. meses.popitem()
```



Tipos de datos

Diccionario

```
1. numero_meses = {2:'mes dos', 4:'mes cuatro'}
2. meses.update(numero_meses)
3. nombre_meses = [(2, 'febrero'), (4, 'abril')]
4. meses.update(nombre_meses)
```

```
1. numero_meses = {2:'mes dos', 4:'mes cuatro'}
2. meses.update(numero_meses)
3. nombre_meses = [(2, 'febrero'), (4, 'abril')]
4. meses.update(nombre_meses)
```

```
1. diccionario = {}.fromkeys(range(1,11))
2. diccionario = {}.fromkeys(range(1,11),0)
```

```
1. meses.clear()
```

Variables

 Python Variables	
<code>x = 45</code>	Type = Integer  <code>45</code> x
<code>name = "DataFlair"</code>	Type = String <code>"DataFlair"</code> name
<code>nums = [1, 8.5, 9]</code>	Type = Lists <code>[1, 8.5, 9]</code> nums

Las variables en Python son locales por defecto. Esto quiere decir que las variables definidas y utilizadas en el bloque de código de una *función*, sólo tienen existencia dentro de la misma, y no interfieren con otras variables del resto del código.



Variables

```
>>> c = "Hola Mundo" # cadenas de caracteres
>>> type(c) # comprobar tipo de dato
<type 'str'>
>>> e = 23 # número entero
>>> type(e) # comprobar tipo de dato
<type 'int'>
```

```
String c = "Hola Mundo";
int e = 23;
```

```
print("This program show how to comment in python : ")
# This is single line comment
'''
This
is
multiple
line
comment

'''
print("Single And Multiple line comment will not display : ")
```



Variables

```
>>> c = "Hola Plone" # cadenas de caracteres
>>> c
'Hola Plone'
```

```
>>> a, b, c = 5, 3.2, "Hola"
>>> print a
5
>>> print b
3.2
>>> print c
'Hola'
```

```
>>> x = y = z = True
>>> print x
True
>>> print y
True
>>> print z
True
```

Variables



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



Variables

Algunas reglas y convenciones de nombres para las variables y constantes:

- Nunca use símbolos especiales como !, @, #, \$, %, etc.
- El primer carácter no puede ser un número o dígito.
- Las constantes son colocadas dentro de módulos Python y significa que no puede ser cambiado.
- Los nombres de constante y variable debería tener la combinación de letras en minúsculas (de *a* a la *z*) o MAYÚSCULAS (de la *A* a la *Z*) o dígitos (del 0 al 9) o un underscore (`_`). Por ejemplo:
 - `snake_case`
 - `MACRO_CASE`
 - `camelCase`
 - `CapWords`
- Los nombres que comienzan con guión bajo (simple `_` o doble `__`) se reservan para variables con significado especial
- No pueden usarse como identificadores, las palabras reservadas .



Variables

```
>>> import keyword
>>> keyword.kwlist
['and', 'as', 'assert', 'break', 'class', 'continue', 'def',
'del', 'elif', 'else', 'except', 'exec', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'not',
'or', 'pass', 'print', 'raise', 'return', 'try', 'while',
'with', 'yield']
```

```
>>> import keyword
>>> keyword.iskeyword('as')
True
>>> keyword.iskeyword('x')
False
```



Variables

La sentencia `del` se define recursivamente muy similar a la forma en el cual se define la asignación. A continuación unos ejemplos donde se inicializan variables:

```
>>> cadena, numero, lista = "Hola Plone", 123456, [7,8,9,0]
>>> tupla = (11, "Chao Plone", True, None)
>>> diccionario = {"nombre":"Leonardo","apellido":"Caballero"}
```

```
>>> vars()
{'tupla': (11, 'Chao Plone', True, None),
 '__builtins__': <module '__builtin__' (built-in)>,
 'numero': 123456, '__package__': None, 'cadena': 'Hola Plone',
 'diccionario': {'apellido': 'Caballero', 'nombre': 'Leonardo'},
 '__name__': '__main__', 'lista': [7, 8, 9, 0], '__doc__': None}
```

```
>>> del cadena
>>> vars()
{'tupla': (11, 'Chao Plone', True, None),
 '__builtins__': <module '__builtin__' (built-in)>,
 'numero': 123456, '__package__': None,
 'diccionario': {'apellido': 'Caballero', 'nombre': 'Leonardo'},
 '__name__': '__main__', 'lista': [7, 8, 9, 0], '__doc__': None}
```



Variables

La sentencia `del` se define recursivamente muy similar a la forma en el cual se define la asignación. A continuación unos ejemplos donde se inicializan variables:

```
>>> del numero, lista, tupla, diccionario
>>> vars()
{'__builtins__': <module '__builtin__' (built-in)>,
 '__package__': None, '__name__': '__main__', '__doc__': None}
```

```
>>> variable1 = "variable original"
>>> def variable_global():
...     global variable1
...     variable1 = "variable global modificada"
...
>>> print variable1
variable original
>>> variable_global()
>>> print variable1
variable global modificada
```



Constantes

Las constantes en programación son útiles para dar valores que vamos a utilizar en nuestros programas y que no variaran durante la ejecución del mismo.

En python, no tenemos ninguna palabra reservada que nos permita crear una constante haciendo que esta no se pueda modificar durante el transcurso del programa.

Constantes



```
#!/usr/bin/python

#Variable que indicamos que es una constante
#En este caso, es como una variable mas
IVA = 0.21

precio = 40

precioFinal = precio * (1+IVA)

print(precioFinal)
```



Constantes

```
1  #!/usr/bin/python
2
3  #Aqui crearemos nuestras propias constantes
4  VARIABLE1=50
5  VARIABLE2="Ejemplo"
6  VARIABLE3=25.5
```

```
1  #!/usr/bin/python
2
3  import constantes
4
5  print(constantes.VARIABLE1)
6  print(constantes.VARIABLE2)
7  print(constantes.VARIABLE3)
```



Constantes

Un pequeño número de constantes vive en el espacio de nombres incorporado. Son las siguientes:

`None`

Más información consulte sobre [`None`](#).

`NotImplemented`

Más información consulte sobre [`NotImplemented`](#).

`Ellipsis`

Más información consulte sobre [`Ellipsis`](#).

`False`

El valor falso del tipo [`booleano`](#).

`True`

El valor verdadero del tipo [`booleano`](#).

`__debug__`

Esta constante su valor es `True` si Python no se inició con una opción `-O`. Véase también la sentencia [`assert`](#).

Números



Clase	Tipo	Notas	Ejemplo
<code>int</code>	Números	Número entero con precisión fija.	42
<code>long</code>	Números	Número entero en caso de overflow.	42L ó 456966786151987643L
<code>float</code>	Números	Coma flotante de doble precisión.	3.1415927
<code>complex</code>	Números	Parte real y parte imaginaria j .	(4.5 + 3j)



Números

```
>>> entero = 23
>>> type(entero)
<type 'int'>
```

```
# 027 octal = 23 en base 10
entero = 027
```

```
# 0x17 hexadecimal = 23 en base 10
entero = 0x17
```

```
entero_long = 7L
print entero_long, type(entero_long)
```



Números

```
real = 0.2703
```

```
real = 0.1e-3
```

```
float_1, float_2, float_3 = 0.348, 10.5, 1.5e2  
print float_1, type(float_1)  
print float_2, type(float_2)  
print float_3, type(float_3)
```

```
real = 0.56e-3  
print real, type(real)
```

```
complejo = 2.1 + 7.8j
```

```
complejo = 3.14j  
print complejo, complejo.imag, complejo.real, type(complejo)
```



Números

Para convertir a tipos numéricos debe usar las siguientes funciones integradas en el interprete Python:

- La función int() devuelve un tipo de datos número entero.
- La función long() devuelve un tipo de datos número entero long.
- La función float() devuelve un tipo de datos número entero float.
- La función complex() devuelve un tipo de datos número complejo.

```
>>> help(int)
```

```
bit_length(...)
```

```
int.bit_length() -> int
```

```
Number of bits necessary to represent self in binary.
```

```
>>> bin(37)
```

```
'0b100101'
```

```
>>> (37).bit_length()
```

```
6
```



Números

Clase	Tipo	Notas	Ejemplo
bool	Números	Valor booleano falso.	False
bool	Números	Valor booleano verdadero.	True

En el contexto de las operaciones booleanas, y también cuando las expresiones son usadas bajo sentencias de flujo de control, los siguientes valores son interpretados como False:

- False.
- None.
- Número cero en todos los tipos.
- Cadena de caracteres vacías.
- Contenedores, incluyendo cadenas de caracteres, tuplas, listas, diccionarios y conjuntos mutables e inmutables.



Booleanos

```
>>> False
False
>>> False == False
True
>>> 0 == False
True
>>> "" == False
False
>>> None == False
False
>>> [] == False
False
>>> () == False
False
>>> {} == False
False
>>> ['', ''] == False
False
```



Booleanos

```
>>> int(False)
0
>>> int(True)
1
```

```
>>> type(True)
<type 'bool'>
>>> str(True)
'True'
>>> type(str(True))
<type 'str'>
>>>
>>> type(False)
<type 'bool'>
>>> str(False)
'False'
>>> type(str(False))
<type 'str'>
```



Cadenas

Cortas

Son caracteres encerrado entre comillas simples (') o dobles (").

```
>>> 'Hola Mundo'  
'Hola Mundo'
```

Largas

```
>>> """Clase que representa una Persona"""  
'Clase que representa una Persona'  
>>> '''Clase que representa un Supervisor'''  
'Clase que representa un Supervisor'
```



Cadenas

Clases

`basestring`

`str`

Son *secuencias inmutables* de cadenas de caracteres con soporte a caracteres ASCII.

```
>>> 'Hola Mundo'
'Hola Mundo'
>>> "Hola Mundo"
'Hola Mundo'
```



Cadenas

Clases

`basestring`

¿Qué es Unicode?

Unicode es un estándar universal de codificación de caracteres que se utiliza para admitir caracteres no compatibles con ASCII. El Internet fue desarrollado originalmente en caracteres [ASCII](#), que se basa en el alfabeto inglés y consta de solamente 128 caracteres.

unicode

Son *secuencias inmutables* de cadenas de caracteres con soporte a caracteres Unicode.

```
>>> u'Jekechitü'  
u'Jekechit\xfc'
```



Prefijo de cadenas

Una cadena puede estar precedida por el carácter:

- r/R, el cual indica, que se trata de una cadena raw (del inglés, cruda). Las cadenas raw se distinguen de las normales en que los caracteres escapados mediante la barra invertida (\) no se sustituyen por sus contrapartidas. Esto es especialmente útil, por ejemplo, para usar las expresiones regulares.

```
>>> raw = r"\t\nHola Plone\n"
>>> type(raw)
<type 'str'>
```

- u/U, el cual indica, que se trata de una cadena que utiliza codificación unicode.

```
>>> saber_mas = u"Atüjaa oo'omüin..."
>>> type(saber_mas)
<type 'unicode'>
>>> vocales = U"äóè"
>>> type(vocales)
<type 'unicode'>
```



Cadenas de escape

Para escapar caracteres dentro de cadenas de caracteres se usa el carácter `\` seguido de cualquier carácter ASCII.

Secuencia Escape	Significado
<code>\newline</code>	Ignorado
<code>\\</code>	Backslash (<code>\</code>)
<code>\'</code>	Comillas simple (<code>'</code>)
<code>\"</code>	Comillas doble (<code>"</code>)
<code>\a</code>	Bell ASCII (BEL)
<code>\b</code>	Backspace ASCII (BS)
<code>\f</code>	Formfeed ASCII (FF)
<code>\n</code>	Linefeed ASCII (LF)
<code>\N{name}</code>	Carácter llamado <i>name</i> en base de datos Unicode (Solo Unicode)
<code>\r</code>	Carriage Return ASCII (CR)
<code>\t</code>	Tabulación Horizontal ASCII (TAB)
<code>\uxxxx</code>	Carácter con valor hex 16-bit <i>xxxx</i> (Solamente Unicode). Ver <u><i>hex</i></u> .
<code>\Uxxxxxxxx</code>	Carácter con valor hex 32-bit <i>xxxxxxxx</i> (Solamente Unicode). Ver <u><i>hex</i></u> .
<code>\v</code>	Tabulación Vertical ASCII (VT)
<code>\ooo</code>	Carácter con valor octal <i>ooo</i> . Ver <u><i>octal</i></u> .
<code>\xhh</code>	Carácter con valor hex <i>hh</i> . Ver <u><i>hex</i></u> .



Cadenas de escape

También es posible encerrar una cadena entre triples comillas (simples o dobles). De esta forma puede escribir el texto en varias líneas, y al imprimir la cadena, se respetarán los saltos de línea que se introdujeron sin tener que recurrir a los caracteres escapados y las comillas como los anteriores.

```
In [104]: multiline="""line 1
...: line 2
...: line 3
...: """
```

```
In [105]: multiline
```

```
Out[105]: 'line 1\nline 2\nline 3\n'
```

```
In [106]: print(multiline)
```

```
line 1
```

```
line 2
```

```
line 3
```



Operaciones con Cadenas

```
>>> a, b = "uno", "dos"  
>>> a + b  
'unodos'
```

```
>>> c = "tres"  
>>> c * 3  
'trestrestres'
```

```
>>> # comentarios en linea  
...  
>>>
```



Operaciones con Cadenas

Docstrings

En Python todos los objetos cuentan con una variable especial llamada `__doc__`, gracias a la cual puede describir para qué sirven los objetos y cómo se usan. Estas variables reciben el nombre de docstrings, o cadenas de documentación.

Ten en cuenta, una buena documentación siempre dará respuesta a las dos preguntas:

- ¿Para qué sirve?
- ¿Cómo se utiliza?

Python implementa un sistema muy sencillo para establecer el valor de las docstrings en las funciones, únicamente tiene que crear un comentario en la primera línea después de la declaración.

```
>>> def hola(arg):  
...     """El docstring de la función"""  
...     print "Hola", arg, "!"  
...  
>>> hola("Plone")  
Hola Plone !
```



Operaciones con Cadenas

Formateo de cadenas

Formateo %

El carácter modulo % es un operador integrado en Python. Ese es conocido como el operador de interpolación. Usted necesitará proveer el % seguido por el tipo que necesita ser formateado o convertido. El operador % entonces substituye la frase '%tipodato' con cero o mas elementos del tipo de datos especificado:

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print "el resultado de %s es %f" % (tipo_calculo, valor)
el resultado de raíz cuadrada de dos es 1.414214
```



Operaciones con Cadenas

Formateo de cadenas

Formateo %

Con esta sintaxis hay que determinar el tipo del objeto:

- `%c` = str, simple carácter.
- `%s` = str, cadena de carácter.
- `%d` = int, enteros.
- `%f` = float, coma flotante.
- `%o` = octal.
- `%x` = hexadecimal.



Operaciones con Cadenas

Formateo de cadenas

Formateo %

```
>>> print "CMS: %s, ¿Activar S o N?: %c" % ("Plone", "S")
CMS: Plone, ¿Activar S o N?: S
>>> print "N. factura: %d, Total a pagar: %f" % (345, 658.23)
N. factura: 345, Total a pagar: 658.230000
>>> print "Tipo Octal: %o, Tipo Hexadecimal: %x" % (027, 0x17)
Tipo Octal: 27, Tipo Hexadecimal: 17
```



Operaciones con Cadenas

Formateo de cadenas

`format()`

Este método devuelve una versión formateada de una cadena de caracteres, usando substituciones desde argumentos **args** y **kwargs**. Las substituciones son identificadas entre llaves { } dentro de la cadena de caracteres (llamados campos de formato), y son sustituidos en el orden con que aparecen como argumentos de **format()**, contando a partir de cero (*argumentos posicionales*).

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print "el resultado de {} es {}".format(tipo_calculo, valor)
el resultado de raíz cuadrada de dos es 1.41421356237
```



Operaciones con Cadenas

Formateo de cadenas

`format()`

También se puede referenciar a partir de la posición de los valores utilizando índices:

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print "el resultado de {0} es {1}".format(tipo_calculo, valor)
el resultado de raíz cuadrada de dos es 1.41421356237
```

Los objetos también pueden ser referenciados utilizando un identificador con una clave y luego pasarla como argumento al método:

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> print "el resultado de {nombre} es {resultado}".format(
...     nombre=tipo_calculo, resultado=2**0.5)
el resultado de raíz cuadrada de dos es 1.41421356237
```



Operaciones con Cadenas

Formateo avanzado

Alinear una cadena de caracteres a la derecha en 30 caracteres, con la siguiente sentencia:

```
>>> print "{:>30}".format("raíz cuadrada de dos")
raíz cuadrada de dos
```

Alinear una cadena de caracteres a la izquierda en 30 caracteres (crea espacios a la derecha), con la siguiente sentencia:

```
>>> print "{:30}".format("raíz cuadrada de dos")
raíz cuadrada de dos
```

Alinear una cadena de caracteres al centro en 30 caracteres, con la siguiente sentencia:

```
>>> print "{:^30}".format("raíz cuadrada de dos")
raíz cuadrada de dos
```

Activar Winc



Operaciones con Cadenas

Formateo avanzado

Truncamiento a 9 caracteres, con la siguiente sentencia:

```
>>> print "{:.9}".format("raíz cuadrada de dos")  
raíz cua
```

Alinear una cadena de caracteres a la derecha en 30 caracteres con truncamiento de 9, con la siguiente sentencia:

```
>>> print "{:>30.9}".format("raíz cuadrada de dos")  
raíz cua
```



Operaciones con Cadenas

Formateo por tipo

Opcionalmente se puede poner el signo de dos puntos después del número o nombre, y explicitar el tipo del objeto:

- `s` para cadenas de caracteres (tipo `str`).
- `d` para números enteros (tipo `int`).
- `f` para números de coma flotante (tipo `float`).

Esto permite controlar el formato de impresión del objeto. Por ejemplo, usted puede utilizar la expresión `.4f` para determinar que un número de coma flotante (`f`) se imprima con cuatro dígitos después de la coma (`.4`).

```
>>> tipo_calculo = "raíz cuadrada de dos"
>>> valor = 2**0.5
>>> print "el resultado de {0} es {resultado:.4f}".format(
...     tipo_calculo, resultado=valor)
el resultado de raíz cuadrada de dos es 1.4142
```



Operaciones con Cadenas

Formateo por tipo

Formateo de números enteros, rellenos con espacios, con las siguientes sentencias:

```
>>> print "{:4d}".format(10)
    10
>>> print "{:4d}".format(100)
   100
>>> print "{:4d}".format(1000)
  1000
```

Formateo de números enteros, rellenos con ceros, con las siguientes sentencias:

```
>>> print "{:04d}".format(10)
0010
>>> print "{:04d}".format(100)
0100
>>> print "{:04d}".format(1000)
1000
```



Operaciones con Cadenas

Formateo por tipo

Formateo de números flotantes, rellenos con espacios, con las siguientes sentencias:

```
>>> print "{:7.3f}".format(3.1415926)
      3.142
>>> print "{:7.3f}".format(153.21)
    153.210
```

Formateo de números flotantes, rellenos con ceros, con las siguientes sentencias:

```
>>> print "{:07.3f}".format(3.1415926)
003.142
>>> print "{:07.3f}".format(153.21)
153.210
```



Operadores de asignaciones

Los operadores de asignación se utilizan para

Existe en Python todo un grupo de operadores los cuales le permiten básicamente asignar un valor a una variable, usando el operador “=”. Con estos operadores pueden aplicar la técnica denominada asignación aumentada.

```
a, b, c = 21, 10, 0
```

```
print "Valor de variable 'a':", a
```

```
print "Valor de variable 'b':", b
```



Operadores de asignaciones

```
c = a + b
```

```
c += a
```

```
c *= a
```

```
c /= a
```

```
c = 2
```

```
c %= a
```

```
c **= a
```

```
c //= a
```



Operadores aritméticos

El operador `+` suma los valores de tipo de datos numéricos.

```
>>> 3 + 2  
5
```

El operador `-` resta los valores de tipo de datos numéricos.

```
>>> 4 - 7  
-3
```

El operador `-` asigna un valor negativo a un tipo de datos numéricos.

```
>>> -7  
-7
```

El operador `*` multiplica los valores de tipo de datos numéricos.

```
>>> 2 * 6  
12
```



Operadores aritméticos

El operador `**` calcula el exponente entre valores de tipo de datos numéricos.

```
>>> 2 ** 6  
64
```

El operador *división* el resultado que se devuelve es un número real.

```
>>> 3.5 / 2  
1.75
```

El operador *división entera* el resultado que se devuelve es solo la parte entera.

```
>>> 3.5 // 2  
1.0
```

No obstante hay que tener en cuenta que si utilizamos dos operandos enteros, Python determinará que quiere que la variable resultado también sea un entero, por lo que el resultado de, por ejemplo, `3 / 2` y `3 // 2` sería el mismo: **1**.



Operadores aritméticos

Si quisiéramos obtener los decimales necesitaríamos que al menos uno de los operandos fuera un número real, bien indicando los decimales:

```
r = 3.0 / 2
```

O bien utilizando la función `float()` para convertir a entero coma flotante o real:

```
r = float(3) / 2
```

Esto es así porque cuando se mezclan tipos de números, Python convierte todos los operandos al tipo más complejo de entre los tipos de los operandos.



Operadores aritméticos

Si quisiéramos obtener los decimales necesitaríamos que al menos uno de los operandos fuera un número real, bien indicando los decimales:

```
r = 3.0 / 2
```

O bien utilizando la función `float()` para convertir a entero coma flotante o real:

```
r = float(3) / 2
```

Esto es así porque cuando se mezclan tipos de números, Python convierte todos los operandos al tipo más complejo de entre los tipos de los operandos.

El operador *módulo* no hace otra cosa que devolver el resto de la división entre los dos operandos. En el ejemplo, `7 / 2` sería `3`, con `1` de resto, luego el *módulo* es `1`.

```
>>> 7 % 2  
1
```



Orden de precedencia

El orden de precedencia de ejecución de los operadores aritméticos es:

1. Exponente: `**`
2. Negación: `-`
3. Multiplicación, División, División entera, Módulo: `*`, `/`, `//`, `%`
4. Suma, Resta: `+`, `-`

Eso quiere decir que se debe usar así:

```
>>> 2**1/12
0.16666666666666666
>>>
```



Orden de precedencia

Más igualmente usted puede omitir este orden de precedencia de ejecución de los operadores aritméticos usando paréntesis () anidados entre cada nivel calculo, por ejemplo:

```
>>> 2**(1/12)
1.0594630943592953
>>>
```



Operadores relacionales

Operador ==

El operador `==` evalúa que los valores sean *iguales* para varios tipos de datos.

```
>>> 5 == 3
False
>>> 5 == 5
True
>>> "Plone" == 5
False
>>> "Plone" == "Plone"
True
>>> type("Plone") == str
True
```



Operadores relacionales

Operador !=

El operador `!=` evalúa si los valores son *distintos*.

```
>>> 5 != 3
True
>>> "Plone" != 5
True
>>> "Plone" != False
True
```



Operadores relacionales

Operador <

El operador < evalúa si el valor del lado izquierdo es *menor que* el valor del lado derecho.

```
>>> 5 < 3  
False
```

Operador >

El operador > evalúa si el valor del lado izquierdo es *mayor que* el valor del lado derecho.

```
>>> 5 > 3  
True
```



Operadores relacionales

Operador $\leq$

El operador $\leq$ evalúa si el valor del lado izquierdo es *menor o igual que* el valor del lado derecho.

```
>>> 5 <= 3  
False
```

Operador $\geq$

El operador $\geq$ evalúa si el valor del lado izquierdo es *mayor o igual que* el valor del lado derecho.

```
>>> 5 >= 3  
True
```



Operadores relacionales

```
a, b, a1, b1, c1 = 5, 5, 7, 3, 3
cadena1, cadena2 = 'Hola', 'Adiós'
lista1, lista2 = [1, 'Lista Python', 23], [11, 'Lista Python', 23]
```

```
c = a == b
print c

cadenas = cadena1 == cadena2
print cadenas

listas = lista1 == lista2
print listas
```

```
d = a1 != b
print d

cadena0 = cadena1 != cadena2
print cadena0
```



Operadores relacionales

```
f = b1 < a1  
print f
```

```
e = a1 > b1  
print e
```

```
h = b1 <= c1  
print h
```

```
g = b1 >= c1  
print g
```



Operadores relacionales

```
f = b1 < a1  
print f
```

```
e = a1 > b1  
print e
```

```
h = b1 <= c1  
print h
```

```
g = b1 >= c1  
print g
```



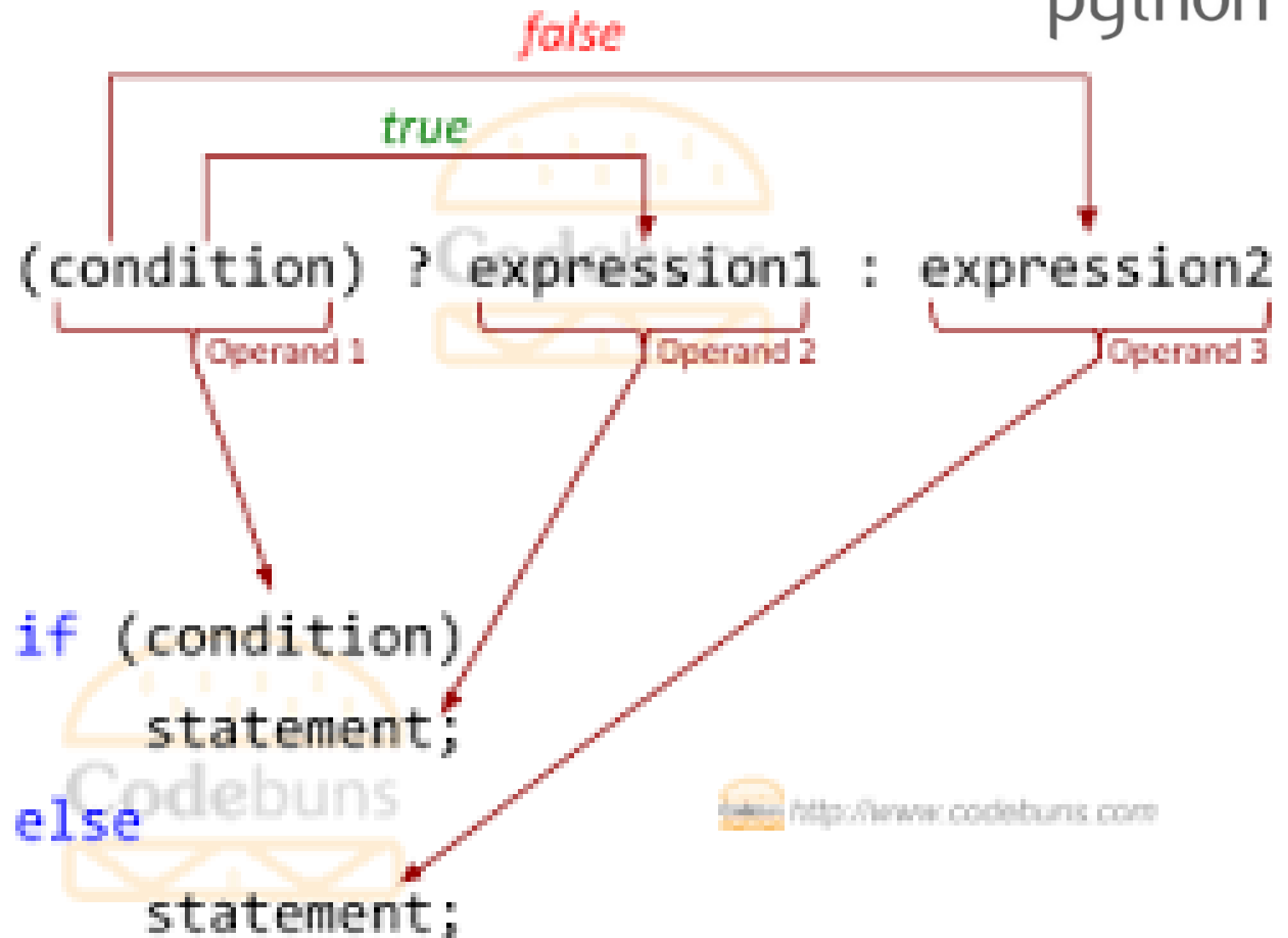
Operadores if

```
a, b = 10, 20
```

```
if (a and b):  
    print "Las variables 'a' y 'b' son VERDADERO."  
else:  
    print "O bien la variable 'a' no es VERDADERO " + \  
        "o la variable 'b' no es VERDADERO."
```

Operadores if

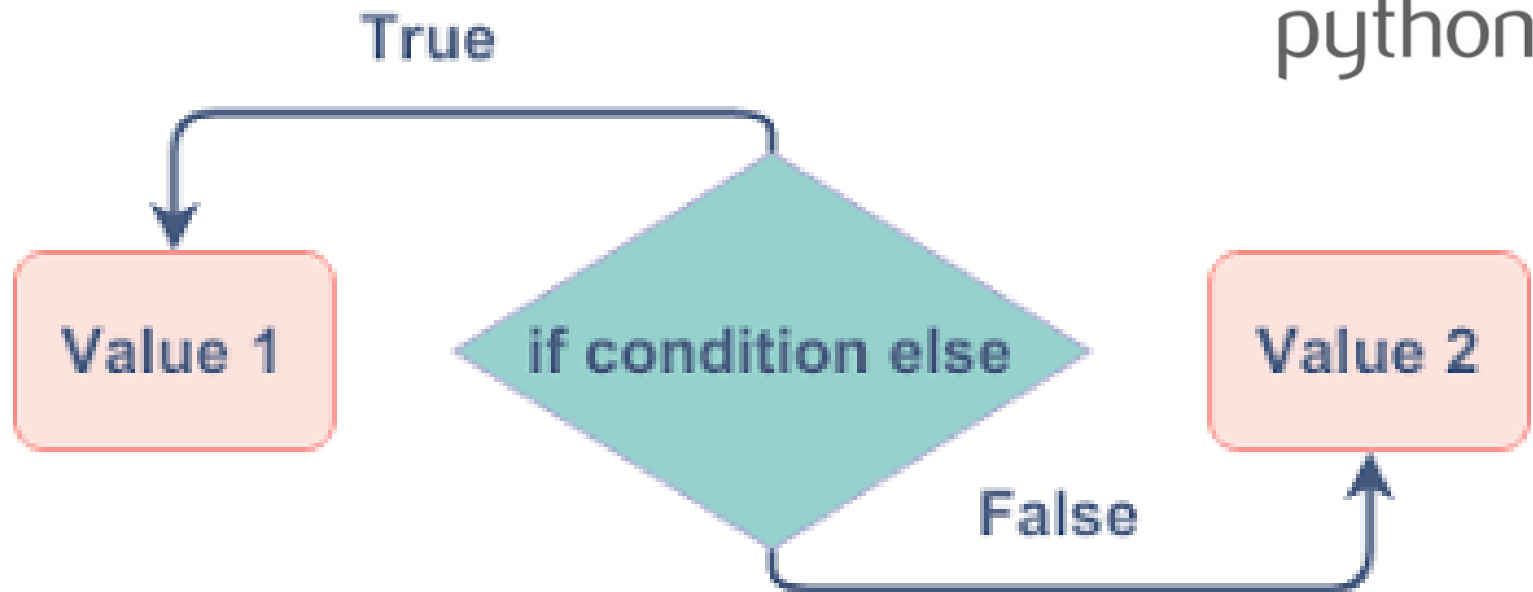
Ternary





Operadores if

Ternary



```
# Program to demonstrate conditional operator
```

```
a, b = 10, 20
```

```
# Copy value of a in min if a < b else copy b
```

```
min = a if a < b else b
```

```
print(min)
```



Operadores if

Ternary

```
# Normal way
check = False
value = ""
if check == True:
    value = "True"
else:
    value = "False"
```

```
# The python idiom
# use ternary operator
```

```
value = "True" if check else "False"
```

```
# Another way
check = False
value = "False"
if check == True:
    value = "True"
```

```
# or simplified
check = False
value = "False"
if check:
    value = True
```



Operadores while

```
suma, numero = 0, 1

while numero <= 10:
    suma = numero + suma
    numero = numero + 1
print "La suma es " + str(suma)
```

```
variable = 10

while variable > 0:
    print 'Actual valor de variable:', variable
    variable = variable -1
    if variable == 5:
        break
```



Operadores while

```
variable = 10

while variable > 0:
    variable = variable -1
    if variable == 5:
        continue
    print 'Actual valor de variable:', variable
```



Operadores for

```
animales = ['gato', 'perro', 'serpiente']  
for animal in animales:  
    print "El animal es: {0}, tamaño de palabra es: {1}".format(  
        animal, len(animal))
```

```
oracion = 'Mary entiende muy bien Python'  
frases = oracion.split() # convierte a una lista cada palabra  
print "La oración analizada es:", oracion, ".\n"  
for palabra in range(len(frases)):  
    print "Palabra: {0}, en la frase su posición es: {1}".format(  
        frases[palabra], palabra)
```



Operadores for

```
conexion_bd = "127.0.0.1","root","123456","nomina"  
for parametro in conexion_bd:  
    print parametro
```

```
datos_basicos = {  
    "nombres":"Leonardo Jose",  
    "apellidos":"Caballero Garcia",  
    "cedula":"26938401",  
    "fecha_nacimiento":"03/12/1980",  
    "lugar_nacimiento":"Maracaibo, Zulia, Venezuela",  
    "nacionalidad":"Venezolana",  
    "estado_civil":"Soltero"  
}  
clave = datos_basicos.keys()  
valor = datos_basicos.values()  
cantidad_datos = datos_basicos.items()  
  
for clave, valor in cantidad_datos:  
    print clave, ":", valor
```



Manejo de errores

```
try:
    You do your operations here;
    .....
except ExceptionI:
    If there is ExceptionI, then execute this block.
except ExceptionII:
    If there is ExceptionII, then execute this block.
    .....
else:
    If there is no exception then execute this block.
```



Manejo de errores

```
print(x)
```

```
Traceback (most recent call last):  
  File "demo_try_except_error.py", line 3, in <module>  
    print(x)  
NameError: name 'x' is not defined
```

```
try:  
    print(x)  
except:  
    print("An exception occurred")
```



Manejo de errores

```
try:
    print(a) #cambiar por print('a'+2)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
```



Manejo de errores

```
try:
    print('hola')    #cambiar por prin(a) y luego por print('a'+2)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")|
```

```
try:
    print('hola')    #cambiar por prin(a)
except:
    print("Something went wrong")
finally:
    print("Fin")
```



Manejo de errores

```
x = -1

if x < 0:
    raise Exception("Sorry, no numbers below zero")

x = "hello"

if not type(x) is int:
    raise TypeError("Only integers are allowed")
```

Manejo de errores

```
locals()['__builtins__']
```



python™

Sr.No.	Exception Name & Description
1	Exception Base class for all exceptions
2	StopIteration Raised when the next() method of an iterator does not point to any object.
3	SystemExit Raised by the sys.exit() function.
4	StandardError Base class for all built-in exceptions except StopIteration and SystemExit.
5	ArithmeticError Base class for all errors that occur for numeric calculation.
6	OverflowError Raised when a calculation exceeds maximum limit for a numeric type.
7	FloatingPointError Raised when a floating point calculation fails.



python™

Manejo de errores

Sr.No.	Exception Name & Description
8	ZeroDivisionError Raised when division or modulo by zero takes place for all numeric types.
9	AssertionError Raised in case of failure of the Assert statement.
10	AttributeError Raised in case of failure of attribute reference or assignment.
11	EOFError Raised when there is no input from either the <code>raw_input()</code> or <code>input()</code> function and the end of file is reached.
12	ImportError Raised when an import statement fails.
13	KeyboardInterrupt Raised when the user interrupts program execution, usually by pressing Ctrl+c.
14	LookupError Base class for all lookup errors.



python™

Manejo de errores

Sr.No.	Exception Name & Description
15	IndexError Raised when an index is not found in a sequence.
16	KeyError Raised when the specified key is not found in the dictionary.
17	NameError Raised when an identifier is not found in the local or global namespace.
18	UnboundLocalError Raised when trying to access a local variable in a function or method but no value has been assigned to it.
19	EnvironmentError Base class for all exceptions that occur outside the Python environment.
20	IOError Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
21	OSError Raised for operating system-related errors.



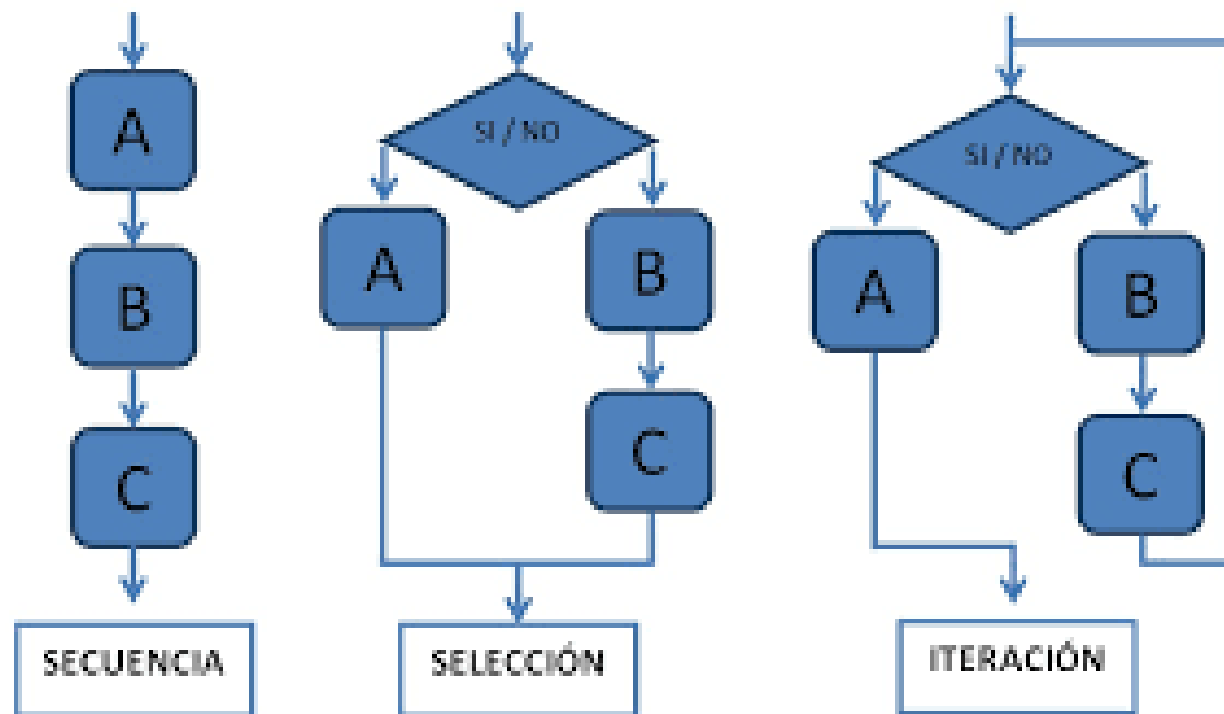
python™

Manejo de errores

Sr.No.	Exception Name & Description
22	SyntaxError Raised when there is an error in Python syntax.
23	IndentationError Raised when indentation is not specified properly.
24	SystemError Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
25	SystemExit Raised when Python interpreter is quit by using the <code>sys.exit()</code> function. If not handled in the code, causes the interpreter to exit.
26	TypeError Raised when an operation or function is attempted that is invalid for the specified data type.
27	ValueError Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
28	RuntimeError Raised when a generated error does not fall into any category.
29	NotImplementedError Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

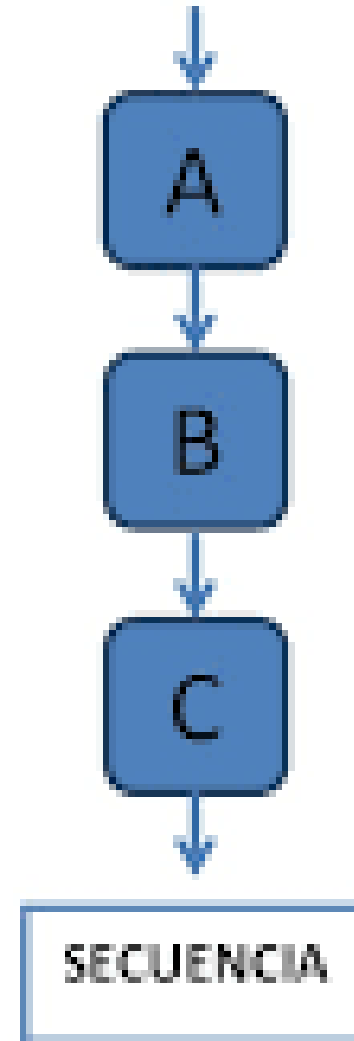
Funciones y programación estructurada

La programación estructurada es un paradigma de programación basado en utilizar funciones o subrutinas, y únicamente tres estructuras de control:



Funciones y programación estructurada

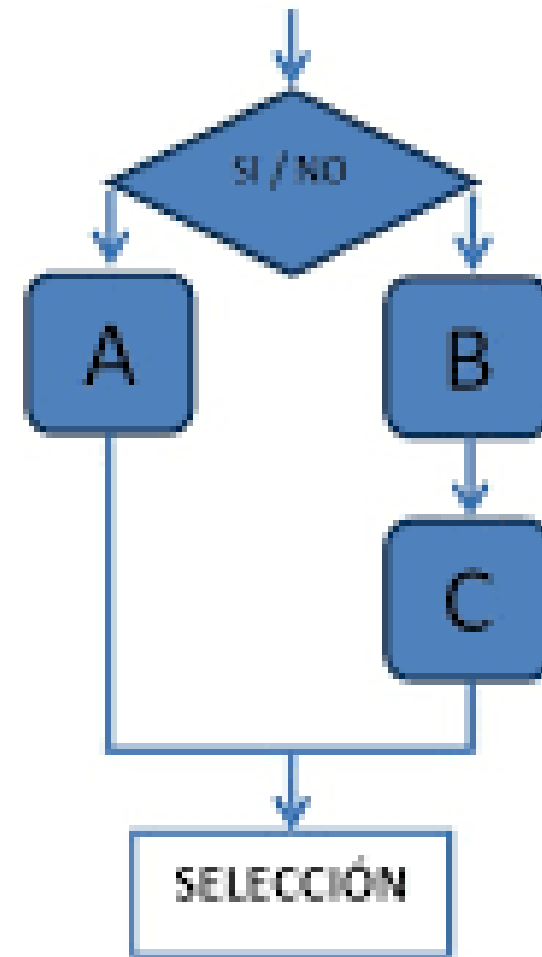
- **Secuencia:** ejecución de una sentencia tras otra.





Funciones y programación estructurada

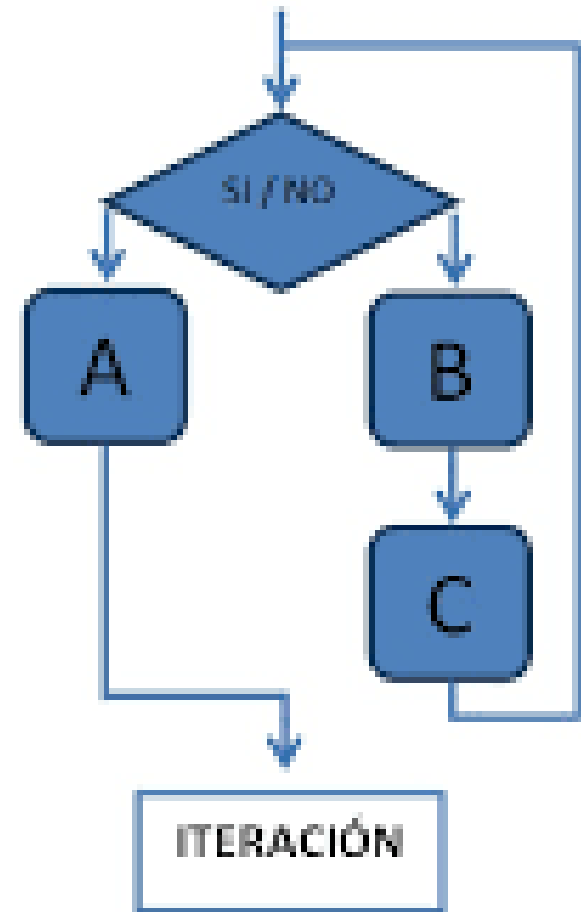
Selección o condicional:
ejecución de una
sentencia o conjunto de
sentencias, según el valor
de una variable
booleana..





Funciones y programación estructurada

Iteración (ciclo o bucle):
ejecución de una
sentencia o conjunto de
sentencias, mientras una
variable booleana sea
verdadera.

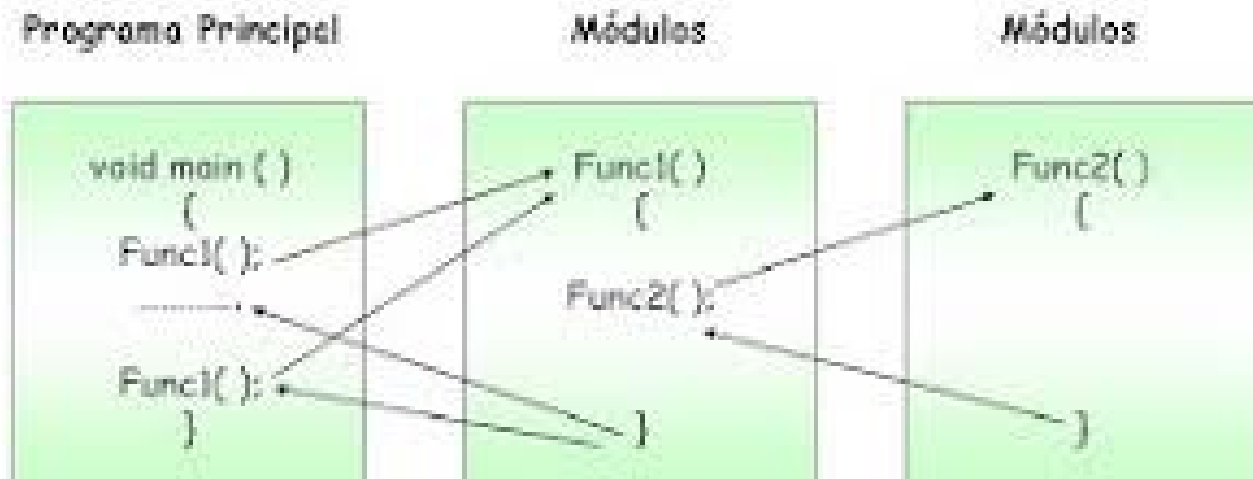




Funciones y programación estructurada

funciones

son bloques de código fuente que pueden contener sentencias reusables de código que puede ser personalizables vía parámetros.





Funciones y programación estructurada

Ventajas

Los programas son más fáciles de entender, pueden ser leídos de forma secuencial y no hay necesidad de tener que rastrear saltos de líneas (GOTO) dentro de los bloques de código para intentar entender la lógica interna.

La estructura de los programas es clara, puesto que las sentencias están más ligadas o relacionadas entre sí.



Funciones y programación estructurada

Ventajas

Se optimiza el esfuerzo en las fases de pruebas y depuración. El seguimiento de los fallos o errores del programa (debugging), y con él su detección y corrección, se facilita enormemente.

Se reducen los costos de mantenimiento. Análogamente a la depuración, durante la fase de mantenimiento, modificar o extender los programas resulta más fácil.



Funciones y programación estructurada

Ventajas

Los programas son más sencillos y más rápidos de confeccionar.

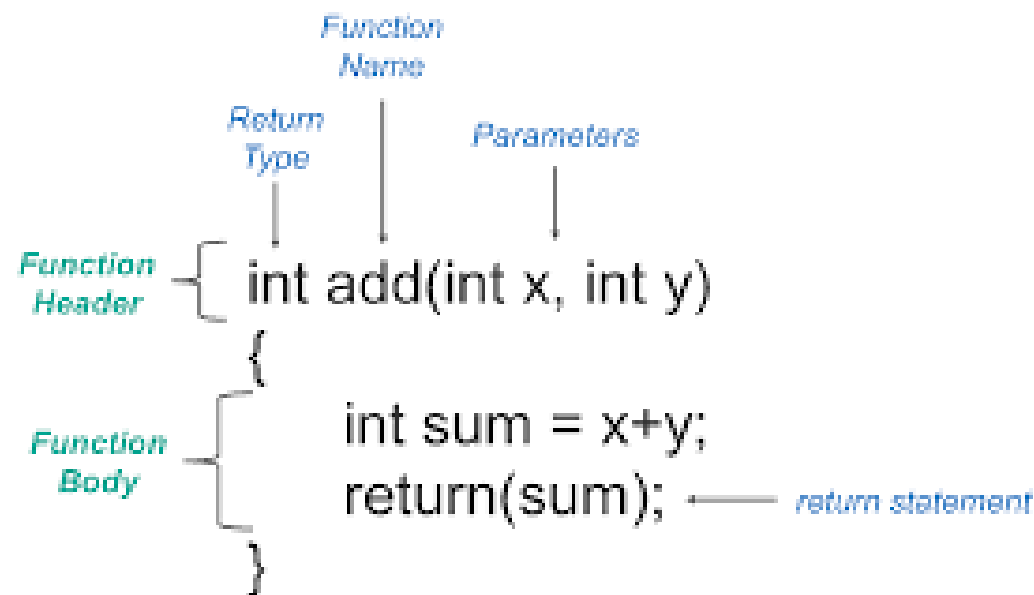
Se incrementa el rendimiento de los programadores



Funciones y programación estructurada

Funciones

Una función es un bloque de código con un nombre asociado, que recibe cero o más argumentos como entrada, sigue una secuencia de sentencias, la cuales ejecuta una operación deseada y devuelve un valor y/o realiza una tarea, este bloque puede ser llamados cuando se necesite.





Funciones y programación estructurada

Funciones

El uso de funciones es un componente muy importante del paradigma de la programación llamada estructurada, y tiene varias ventajas:

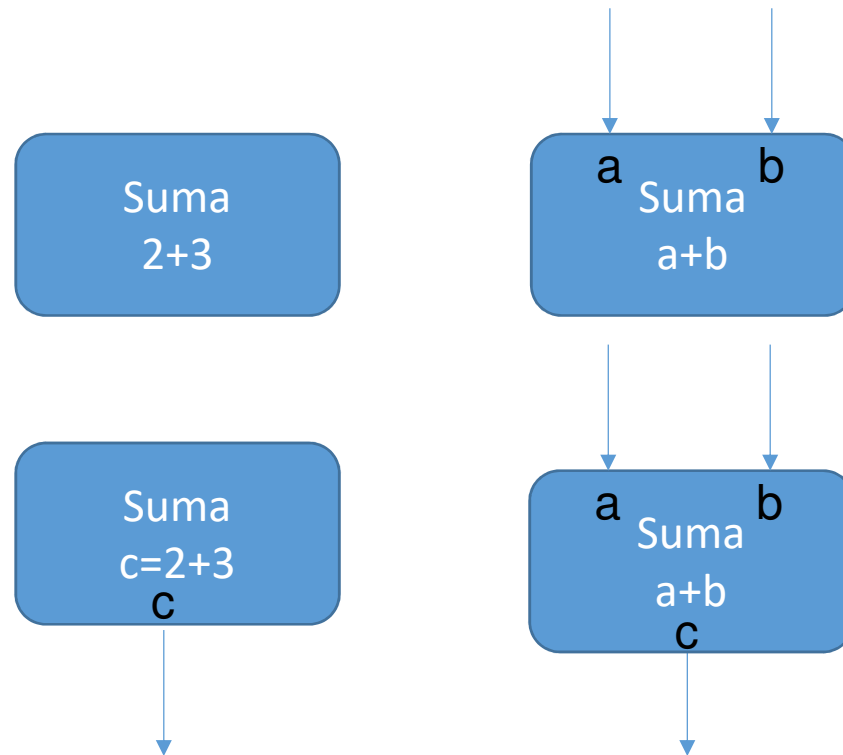
modularización: permite segmentar un programa complejo en una serie de partes o módulos más simples, facilitando así la programación y el depurado.

reutilización: permite reutilizar una misma función en distintos programas.



Funciones y programación estructurada

Funciones





Funciones y programación estructurada

Funciones

Sentencia def

La sentencia **def** es una definición de función usada para crear objetos funciones **definidas por el usuario**.

Una definición de función es una sentencia ejecutable. La definición de función no ejecuta el cuerpo de la función; esto es ejecutado **solamente cuando la función es llamada**.



Funciones y programación estructurada

Sentencia def

```
def NOMBRE(LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

NOMBRE, es el nombre de la función.

LISTA_DE_PARAMETROS, es la lista de parámetros que puede recibir una función.

DOCSTRING_DE_FUNCION, es la cadena de caracteres usada para documentar la función.

SENTENCIAS, es el bloque de sentencias en código fuente Python que realizar cierta operación dada.

RETURN, es la sentencia return en código Python.

EXPRESION, es la expresión o variable que devuelve la sentencia **return**.



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

```
>>> def hola(arg):  
...     """El docstring de la función"""  
...     print "Hola", arg, "!"  
... 
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

```
>>> def resta(a, b):  
...     return a - b  
...  
>>> resta(30, 10)
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

```
>>> def resta(a, b):  
...     return a - b  
...  
>>> resta(b=30, a=10)
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

```
>>> resta()
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

```
>>> resta()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: resta() takes exactly 2 arguments (0 given)
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

```
>>> def resta(a=30 , b=10 ):  
...     return a - b  
...  
>>> resta(30, 10)  
  
>>> resta()
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

```
>>> def p(*args):  
...     for arg in args:  
...         print arg  
...  
>>> p(5, "Hola Plone", [1, 2, 3, 4, 5])
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

```
>>> def nombre(**kwargs):  
...     print kwargs  
...  
>>> nombre(n=5, c="Hola Plone", l=[1,2,3,4,5])
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

```
>>> def suma(numero1,numero2):  
    '''función la cual suma dos números'''  
    return (numero1 + numero2)  
  
>>> suma(23,74)
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

```
>>> def prueba():  
...     return "Plone CMS", 20, [1,2,3]  
...  
>>> prueba()
```



Funciones y programación estructurada

Sentencia def

```
def NOMBRE (LISTA_DE_PARAMETROS):  
    """DOCSTRING_DE_FUNCION"""  
    SENTENCIAS  
    RETURN [EXPRESION]
```

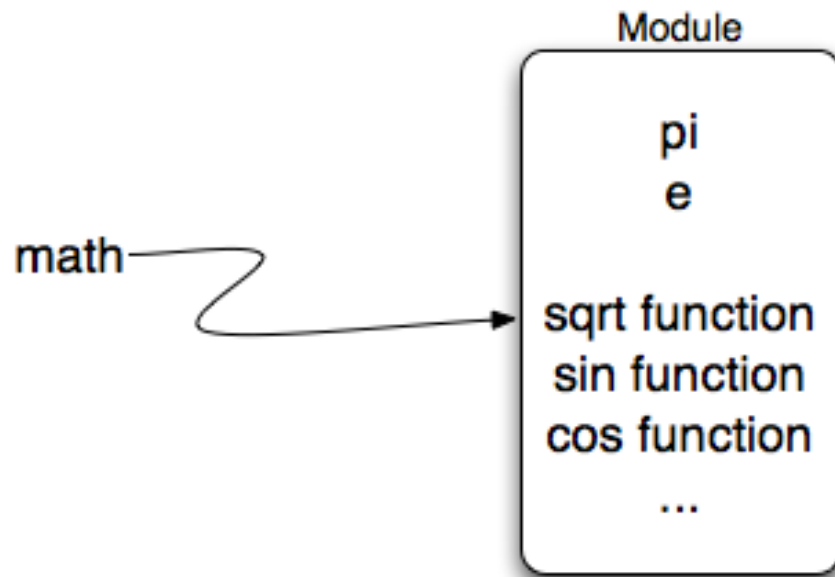
```
>>> cadena, numero, lista = prueba()  
>>> print cadena, type(cadena)  
  
>>> print numero, type(numero)  
...  
>>> print lista, type(lista)
```



Funciones y programación estructurada

Módulos

En Python las diversas aplicaciones Python se encuentran dentro de módulos y paquetes los cuales contienen el sistema de ficheros.



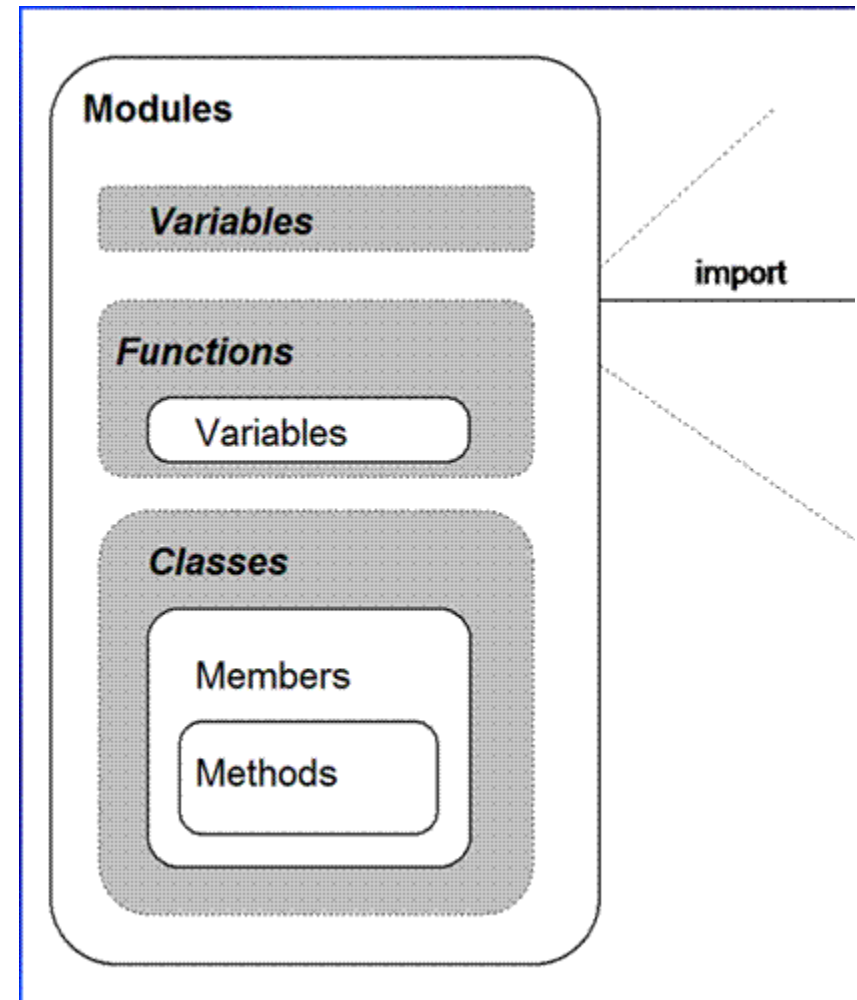
Un módulo es un objeto de Python con atributos con nombres arbitrarios que se pueden enlazar y hacer referencia.



Funciones y programación estructurada

Módulos

Archivo con extensión **.py**.
Un módulo puede definir funciones, clases y variables, también puede incluir código ejecutable





Funciones y programación estructurada

import

importa el módulo si el mismo
esta presente en la ruta de
búsqueda

Una **ruta de búsqueda** es una
lista de directorios que el
interprete busca antes de
importar un módulo.

```
>>> import os  
>>> import re, datetime
```



Funciones y programación estructurada

import

`utilidades.py`

```
""" Módulo para cálculos diversos """  
  
def suma_total(monto=0):  
    """ Calcula la suma total """  
    calculo_suma = 20  
    calculo_suma += monto  
    return calculo_suma
```



Funciones y programación estructurada

import

```
# Importar el modulo llamado "utilidades"  
import utilidades  
  
# Usted puede llamar una función  
# definida dentro del módulo  
print utilidades.suma_total(1,1)  
print dir(utilidades)
```



Funciones y programación estructurada

import

Un módulo **se carga solo una vez**, independientemente de la cantidad de veces que se importe.

La primera vez que un módulo es importado en un script de Python, **se ejecuta su código una vez**. Si otro módulo importa el mismo módulo este no se cargará nuevamente; **los módulos son inicializados una sola vez**.



Funciones y programación estructurada

Localizando módulos

Cuando se importa un módulo, el interprete Python busca por el módulo en la secuencia siguiente:

1. El directorio actual
`os.getcwd()`.
2. Si el módulo no es encontrado, Python entonces busca en cada directorio en la variable de entorno `PYTHONPATH` del sistema operativo.
`os.environ['PYTHONPATH'].split(os.pathsep)`
3. Si todas las anteriores fallan, Python busca la ruta predeterminada. En UNIX, la ruta predeterminada normalmente esta **`/usr/local/lib/python/`**.



Funciones y programación estructurada

Localizando módulos

sys.path

```
utilidades/  
├── utilidades.py  
└── calculo_factura_pipo.py
```



Funciones y programación estructurada

import as

```
import utilidades as util  
  
print(util.suma_total(11))  
|
```



Funciones y programación estructurada

from

```
from utilidades import suma_total  
print suma_total(1,1)
```



Funciones y programación estructurada

from

```
person1 = {  
    "name": "John",  
    "age": 36,  
    "country": "Norway"  
}
```

```
from utilidades import suma_total, person1  
print suma_total(1,1)  
print person1
```



Funciones y programación estructurada

from

```
from utilidades import *
```

```
print suma_total(1,1)
```

```
print suma(1,1)
```



Operaciones de E/S

`raw_input()`

```
>>> nombre = raw_input('Ana: ¿Cómo se llama usted?: ')
Ana: ¿Cómo se llama usted?: Leonardo
>>> print nombre
Leonardo
```

`input()`

```
>>> edad = input('Ana: ¿Que edad tiene usted?: ')
Ana: ¿Que edad tiene usted?: 38
>>> print edad
38
```



Operaciones de E/S

[input\(\)](#)

```
print('suma')
a=input('ingrese el primer valor: ')
b=input('ingrese el segundo valor: ')
c=suma(a,b)
print('la suma es:', c)
```



Operaciones de E/S

Directorios y Archivo

```
import os  
os.mkdir("leo_r")
```

Módulo os

```
>>> import os  
>>> os.listdir("./")
```



Operaciones de E/S

Directorios y Archivo

Módulo os

```
>>> import os
>>> os.path.isfile("leo_r")
```

```
>>> import os
>>> os.path.isdir("leo_r")
```



Operaciones de E/S

Directorios y Archivo

Módulo os

```
>>> import os
>>> os.getcwd()
>>> os.chdir("leo_r"):
>>> os.getcwd()
```

```
>>> os.listdir("./")
>>> os.chdir("../")
>>> os.getcwd()
```



Operaciones de E/S

Directorios y Archivo

Módulo os

```
>>> import os
>>> os.rename("leo_r", "leo_rente")
>>> os.listdir("./")
```

```
>>> os.listdir("./")
>>> os.remove("datos.txt")
>>> os.listdir("./")
```

Operaciones de E/S

Directorios y Archivo

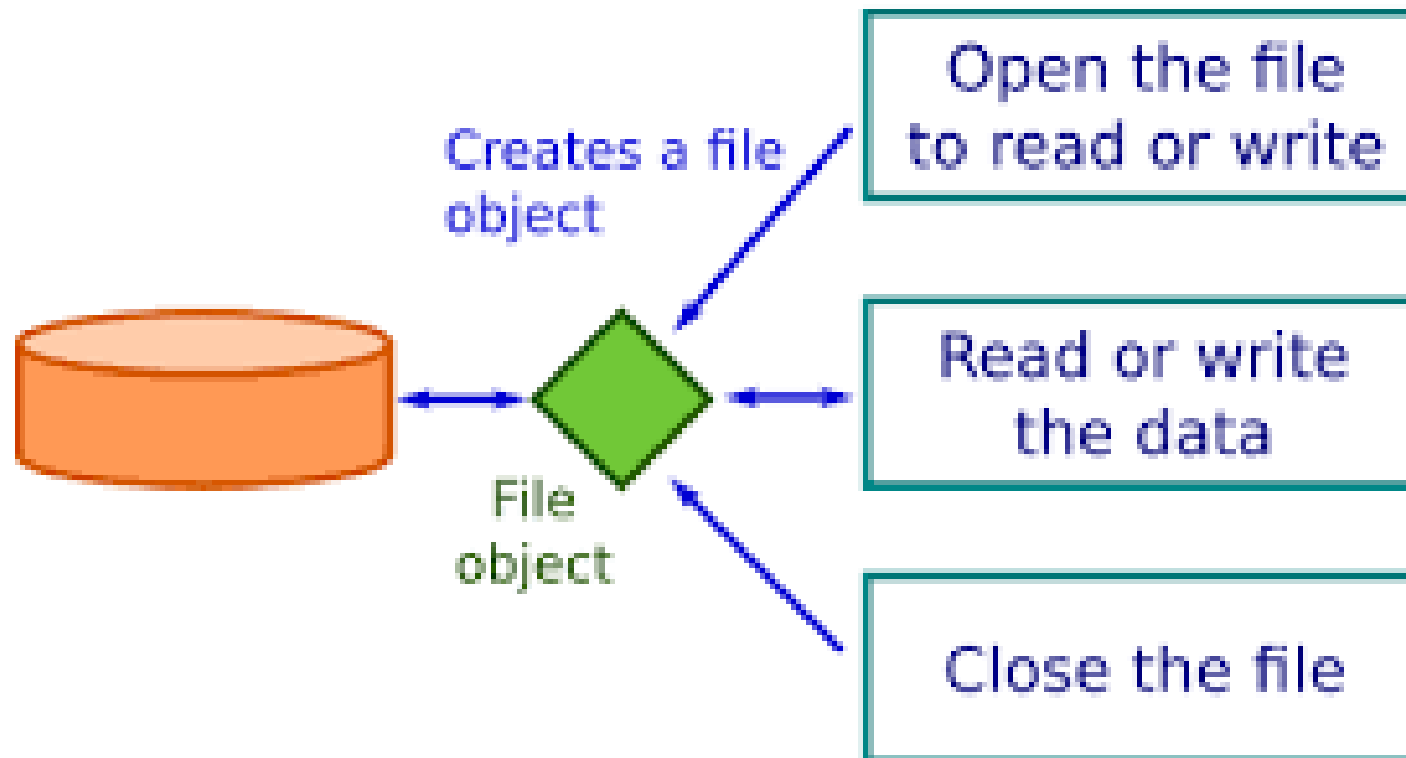


```
>>> os.rmdir("leo_rente")  
>>> os.chdir("leo_rente")
```

Módulo os

Operaciones de E/S

File





Operaciones de E/S

File

```
file object = open(file_name [, access_mode][, buffering])
```

file_name: es un valor de cadena que contiene el nombre del archivo al que desea acceder.

Buffering: Si el valor de se establece en 0, no tiene lugar el almacenamiento en búfer. Si el valor de almacenamiento en búfer es 1, el almacenamiento en línea se realiza al acceder a un archivo

Access_mode: determina el modo en que se debe abrir el archivo, es decir, leer, escribir, agregar, etc. Este es un parámetro opcional y el modo de acceso al archivo predeterminado es read (r).



Operaciones de E/S

File

```
file object = open(file_name [, access_mode][, buffering])
```

Indicador	Modo de apertura	Ubicación del puntero
r	Solo lectura	Al inicio del archivo
rb	Solo lectura en modo binario	Al inicio del archivo
r+	Lectura y escritura	Al inicio del archivo
rb+	Lectura y escritura en modo binario	Al inicio del archivo
w	Solo escritura. Sobrescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
wb	Solo escritura en modo binario. Sobrescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo



Operaciones de E/S

File

```
file object = open(file_name [, access_mode][, buffering])
```

Indicador	Modo de apertura	Ubicación del puntero
w+	Escritura y lectura. Sobrescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
wb+	Escritura y lectura en modo binario. Sobrescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
a	Añadido (agregar contenido). Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
ab	Añadido en modo binario (agregar contenido). Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
a+	Añadido (agregar contenido) y lectura. Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
ab+	Añadido (agregar contenido) y lectura en modo binario. Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo



Operaciones de E/S

File

```
file object = open(file_name [, access_mode][, buffering])
```

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
# Close open file
fo.close()
print "Closed or not : ", fo.closed
```



Operaciones de E/S

File >> *read()*

```
fileObject.read([count])
```

```
# Open a file
fo = open("foo.txt", "r")
str = fo.read(10);
print "Read String is : ", str
str = fo.read(10);
print "Read String is : ", str
# Close opened file
fo.close()
```



Operaciones de E/S

File >> read()

```
fileObject.read( )
```

```
# Open a file
fo = open("foo.txt", "r")
str = fo.read( );
print "Read String is : ", str
str = fo.read( );
print "Read String is : ", str
# Close opened file
fo.close()
```



Operaciones de E/S

File

tell(): Retorna la posición actual del puntero

seek(byte): Mueve el puntero hacia el byte indicado.

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10)
print "Read String is : ", str
```

```
# Check current position
position = fo.tell()
print "Current file position : ", position
```

```
# Reposition pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10)
print "Again read String is : ", str
# Close opened file
fo.close()
```



Operaciones de E/S

File >> readline()

```
fileObject.readline( )
```

```
# Open a file
fo = open("foo.txt", "r")
str = fo.readline( );
print "Read String is : ", str
str = fo.readline( );
print "Read String is : ", str
# Close opened file
fo.close()
```



Operaciones de E/S

File >> *readlines()*

```
fileObject.readlines ( )
```

```
# Open a file
fo = open("foo.txt", "r")
str = fo.readlines( );
print "Read String is : ", str
# Close opened file
fo.close()
```



Operaciones de E/S

File >> *readlines()*

```
fileObject.readlines ( )
```

```
>>> f = open("foo.txt")  
>>> list(f)
```



Operaciones de E/S

File >> *readlines()*

```
fileObject.readlines ( )
```

```
>>> with open('jack_russell.png', 'rb') as byte_reader:
>>>     print(byte_reader.read(1))
>>>     print(byte_reader.read(3))
>>>     print(byte_reader.read(2))
>>>     print(byte_reader.read(1))
>>>     print(byte_reader.read(1))
```



Operaciones de E/S

File >> readlines()

```
fileObject.readlines ( )
```

```
fo = open("foo.txt", "r")
for x in fo:
    print(x)
```



Operaciones de E/S

File >> *write(str)*

```
fileObject.write(string)
```

('\\n') no incluido

```
# Open a file
fo = open("foo.txt", "w")
fo.write( "Python is a great language.")
fo.write( "Yeah its great!!")
# Close opened file
fo.close()
```