# Introducción a Python

FUNCIONES Y CLASES EN PYTHON

JOSÉ ORTIZ BEJAR

# Introducción

- Un lenguaje debería no incluir todo lo que alguien podría requerir alguna vez
- En lugar de eso debería permitir a los desarrolladores expresar toda abstracción que deseen [Steele 1999]
- Definir funciones para crear operaciones de mayor nivel
- Agruparlas en librerías para mantenerlas manejables

# Definición de Funciones

- Define una nueva función utilizando def
- El nombre de los parámetros está entre paréntesis

```
def double(x):
    return x * 2
print double(5)
print double(['basalt', 'granite'])
10
['basalt', 'granite', 'basalt', 'granite']
```

- No se pueden declarar tipos de parámetros

# Valores de regreso

- Termina una función en cualquier momento utilizando return
-

```
def sign(x):
    if x < 0:
        return -1
    if x == 0:
        return 0
    return 1
```

# Valores de regreso

- Las funciones con declaraciones return dispersos entre sí son difícil de entender
- Es necesario leer la función línea por línea para determinar qué hace
- En general:
  - Utiliza returns previos al inicio de la función para manejar casos especiales
  - Y posteriormente un return al final para manejar el caso en general

# Toda función regresa algo

- Las funciones con declaraciones return regresan None (Nada). Y return  sin parámetros regresa None

```
def hola():
    print 'HOLA'

def mundo():
    print 'MUNDO'
    return
```

# Toda función regresa algo

**print hola()**
**print mundo()**
**HOLA**
**None**
**MUNDO**
**None**

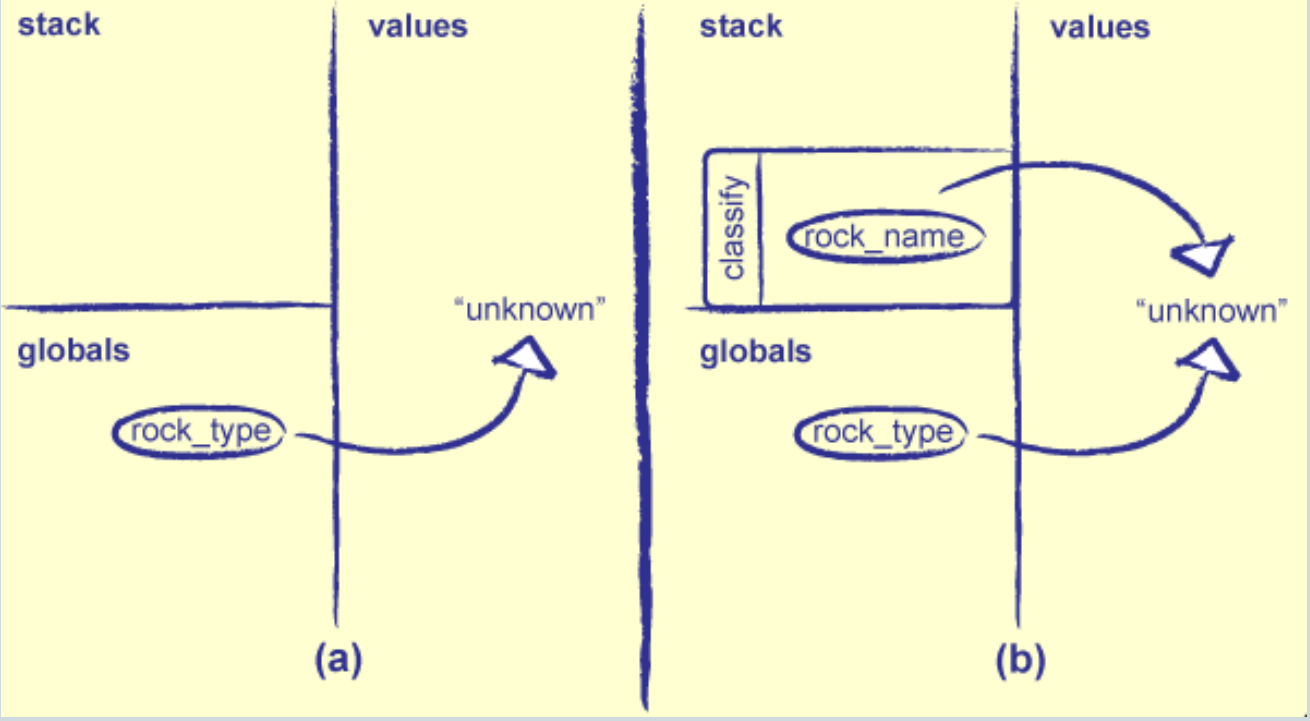
- Entre más coherentes sean las funciones con lo que regresan mejor
- Si una función regresa None, un entero, o una lista, quien invoque a la función tendrá que escribir una declaración if

# Alcance

- Python organiza las variables utilizando call stack

```
# Global variable.
rock_type = 'unknown'

# Function that creates local variable.
def classify(rock_name):
    if rock_name in ['basalt', 'granite']:
        rock_type = 'igneous'
    elif rock_name in ['sandstone', 'shale']:
        rock_type = 'sedimentary'
    else:
        rock_type = 'metamorphic'
    print 'in function, rock_type is', rock_type
```

# Alcance


stack
values
globals
"unknown"
rock_type
(a)
stack
values
classify
rock_name
globals
"unknown"
rock_type
(b)

# Alcance

- Cuando se invoca una función, Python crea un nuevo stack frame
  - Una tabla de pares nombre-/valor
  - Los parámetros son precisamente variables locales que se inicializan automáticamente
- Cuando se hace referencia a una variable, Python la busca en:
  - La cima del stack frame, y después en
  - Las variables globales

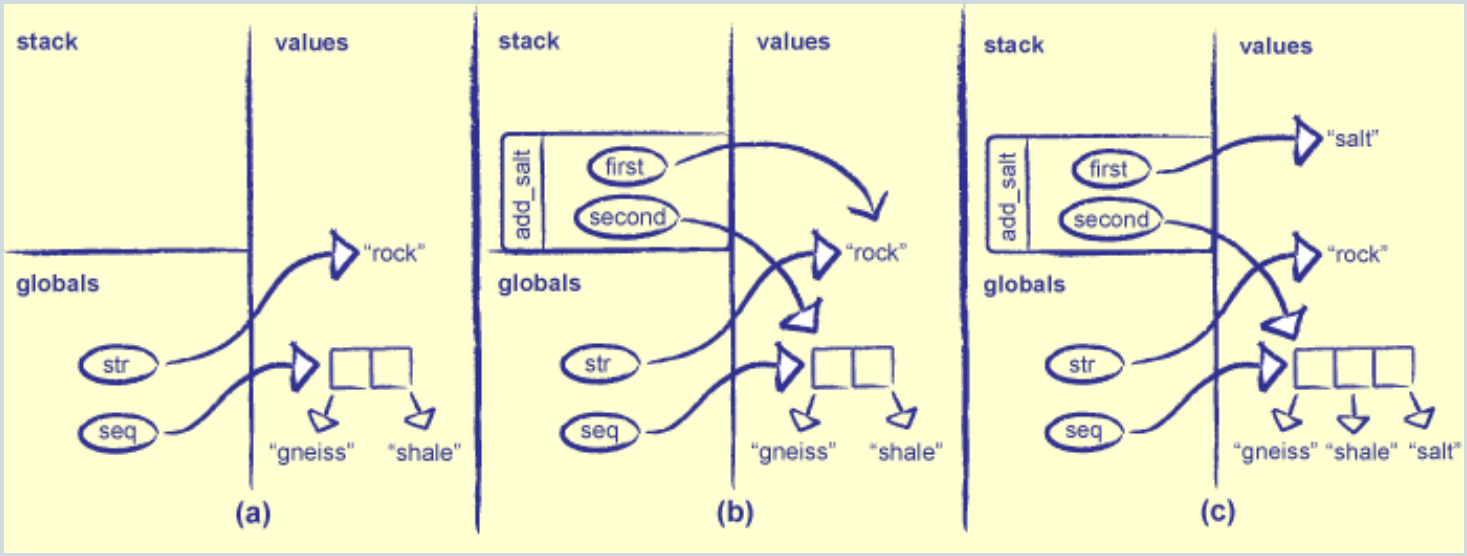
# Reglas de paso de parámetros

- Python copia los valores de las variables cuando se pasan a las funciones
- Pero es importante recordar que las variables contienen referencias a listas
  - Entonces los parámetros son alias
  - Lo cual no es un problema para las cadenas, números y booleanos, ya que permenacen inmutables

# Reglas de paso de parámetros

```
def add_salt(first, second):
    first += "salt"
    second += ["salt"]

str = "rock"
seq = ["gneiss", "shale"]

print "before"
print "str is:", str
print "seq is:", seq

add_salt(str, seq)

print "after"
print "str is:", str
print "seq is:", seq

before
str is: rock
seq is: ['gneiss', 'shale']
after
str is: rock
seq is: ['gneiss', 'shale', 'salt']
```

# Reglas de paso de parámetros



# Elaboración de copias

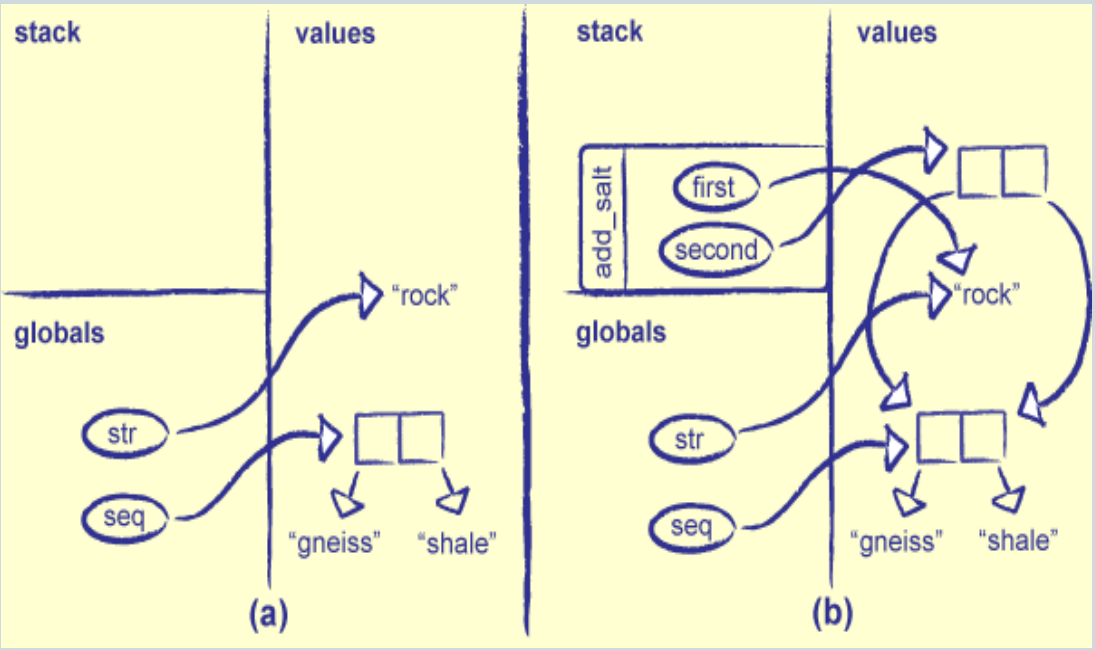
- Para pasar la copia de una lista a una función se requiere dividirla
- values[:] es lo mismo que values[0:len(values)]...
  - ... lo cual es una parte de values que incluye la lista completa...
  - ... y dividir crea una nueva lista

# Elaboración de copias

```
def add_salt(first, second):
    first += "salt"
    second += ["salt"]

str = "rock"
seq = ["gneiss", "shale"]

print "before"
print "str is:", str
print "seq is:", seq

add_salt(str, seq[:])

print "after"
print "str is:", str
print "seq is:", seq
```

```
before
str is: rock
seq is: ['gneiss', 'shale']
after
str is: rock
seq is: ['gneiss', 'shale']
```

# Elaboración de copias



# Valores por defecto

- Al definir una función se pueden especificar valores por defecto para los parámetros
  - Sólo se requiere “asignar” algún valor al parámetro en la definición
- Los parámetros que realmente se pasan cuando se llama a una función son asignados (matched up) de izquierda a derecha
- Todos los parámetros con valores por defecto deben estar después de todos los parámetros sin valores por defecto
  - De lo contrario asignar valores a los parámetros resultaría ambiguo

# Valores por defecto

```
def total(values, start=0, end=None):

    # If no values given, total is zero.
    if not values:
        return 0

    # If no end specified, use the entire sequence.
    if end is None:
        end = len(values)

    # Calculate.
    result = 0
    for i in range(start, end):
        result += values[i]
    return result
```

# Valores por defecto

```
numbers = [10, 20, 30]
print "numbers being added:", numbers
print "total(numbers, 0, 3):", total(numbers, 0, 3)
print "total(numbers, 2):", total(numbers, 2)
print "total(numbers):", total(numbers)
numbers being added: [10, 20, 30]
total(numbers, 0, 3): 60
total(numbers, 2): 30
     total(numbers): 60
```

# Número variable de argumentos

Para definir funciones con un número variable de argumentos colocamos un último parámetro para la función cuyo nombre debe precederse de un signo '*':

```
def varios(param1, param2, *otros):
    for val in otros:
        print otros
```

Es posible pasar los argumentos como un diccionario

```
def suma(x,y,**args):
        if not args:
                return x+y
        else:
                t=0
                for key,val in args.items():
                        t=t+val
                return t+x+y
```

# Las Funciones son objetos

- Las funciones son sólo un objeto más
  - Pasan a ser un objeto que se puede llamar, tal como las cadenas y listas pasan a ser objetos que se pueden indexar
  - def es simplemente una forma breve de "crear una función y asignarla a una variable"

# Las Funciones son objetos

```
def circumference(r):
    return 2 * 3.14159 * r

circ = circumference

print 'circumference(1.0):', circumference(1.0)
print 'circ(2.0):', circ(2.0)
circumference(1.0): 6.28318
circ(2.0): 12.56636
```

# Las Funciones son objetos

- Lo que significa que es posible:
  - Redefinir funciones (tal como es posible reasignar valores a las variables
  - Crear alias para las funciones
  - Pasar funciones como parámetros
  - Almacenar funciones en listas

# Ejemplos de funciones objeto

- Ejemplo: aplica una función a cada valor en una lista

```
def apply_to_list(function, values):
    result = []
    for v in values:
        temp = function(v)
        result.append(temp)
    return result

radii = [0.1, 1.0, 10.0]
print apply_to_list(circumference, radii)
[0.62831800000000004, 6.2831799999999998,
62.831800000000001]
```

# Ejemplos de funciones objeto

- Ejemplo: aplica varias funciones a un solo valor

```
def area(r):
    return 3.14159 * r * r

def color(r):
    return "unknown"

def apply_each(functions, value):
    result = []
    for f in functions:
        temp = f(value)
        result.append(temp)
    return result

functions = [circumference, area, color]
print apply_each(functions, 1.0)

[6.2831799999999998, 3.1415899999999999, 'unknown']
```

# Atributos de funciones

- Toda función tiene un atributo llamado __name__
  - El nombre con el cual fue originalmente definida
  - Práctico a la hora de depurar

```
def sedimentary(rock_name):
    return rock_name in ['sandstone', 'shale']

sed = sedimentary

print 'original name:', sedimentary.__name__
print 'name of alias:', sed.__name__

original name: sedimentary
name of alias: sedimentary
```

# Clases

- En Python las clases se definen mediante la palabra clave class seguida del nombre de la clase, dos puntos (:)
- Igual que para las funciones, si la primera línea del cuerpo se trata de una cadena de texto, esta será la cadena de documentación de la clase o docstring.

# Clases

```
class Cuenta:
    def __init__(self, saldo):
        self.saldo=saldo

    def deposito(self, monto):
        self.saldo+=monto
        print "Su nuevo saldo %s" %self.saldo

    def retiro(self, monto):
        if self.saldo-monto<0:
            print "Saldo insuficiente %s" %self.saldo
        else:
            self.saldo-=monto
            print "Su nuevo saldo %s" %self.saldo
```

# Clases

```
class CuentaInteres(Cuenta):
    def __init__(self):
        Cuenta.__init__(self,1000)
        print "Su saldo de apertura es %s" %self.saldo

    def interes(self):
        self.saldo*=1.01
        print "Su nuevo saldo %f" %self.saldo
```

# Herencia Multiple

```
class Terrestre:
    def desplazar(self):
        print "El animal anda"

class Acuatico:
    def desplazar(self):
        print "El animal nada"

class Cocodrilo(Terrestre, Acuatico):
    pass

c = Cocodrilo()
c.desplazar()
```

# Import

- import es una sentencia
  - Que se ejecuta cuando Python la encuentra, tal como cualquier otra sentencia
- Las sentencias en un módulo se ejecutan mientras éste se carga
  - Asignación y def son sentencias
  - También se pueden utilizar condicionales, ciclos y demás
- Agregando las siguientes líneas de código en geology.py

# Import

```
print 'loading geology module'

def rock_type(rock_name):
    if rock_name in ['basalt', 'granite']:
        return 'igneous'
    elif rock_name in ['sandstone', 'shale']:
        return 'sedimentary'
    else:
        return 'metamorphic'

print 'geology module loaded'
```

- Al correr `analysis.py` imprimirá

```
import geology_debug as geology

for r in ['granite', 'gneiss']:
    print r, 'is', geology.rock_type(r)

loading geology module
geology module loaded
granite is igneous
gneiss is metamorphic
```

# Auto identificación

- Dentro de un módulo, __name__ está definido como:
  - El nombre del módulo, si éste se importa
  - O la cadena "__main__", si es el programa principal (main)
- Frecuentemente utilizado para incluir auto-pruebas en el módulo
  - Se ejecutan las auto-pruebas cuando el módulo se corre de la línea de comandos
  - Las pruebas se omiten cuando otro código carga el módulo

# Auto identificación

```
def is_rock(name):
    return name in ['basalt', 'granite', 'sandstone', 'shale']

if __name__ == '__main__':
    tests = [['basalt', True],  ['gingerale', False],
             [12345678, False], ['sandstone', True]]
    for (value, expected) in tests:
        actual = is_rock(value)
        if actual == expected:
            print 'pass'
        else:
            print 'fail'

$ python self_test.py

pass
pass
pass
pass

$ python
>>> import self_test
>>> self_test.is_rock('sugar')

False
```

# Sistema de librerias

- La librería más comúnmente utilizada en Python es la del sistema sys
  - Información acerca del intérprete de Python (tal como el número de versión y copyright)
  - Información acerca del ambiente (tal como el sistema operativo donde corre el programa)
  - Características avanzadas que simples mortales nunca deberían manipular

# Sistema de librerias

| Tipo | Nombre | Propósito | Ejemplo | Resultado |
|---|---|---|---|---|
| Datos | argv | Los argumentos de la línea de comandos del programa | sys.argv[0] | "myscript.py" (o cualquiera que sea el nombre del programa) |
| | maxint | El valor del número positivo más grande que puede representarel tipo entero básico de Python | sys.maxint | 2147483647 |
| | path | Listado de directorios donde Python busca cuando se importan módulos | sys.path | ['/home/greg/pylib', '/Python24/lib', '/Python24/lib/site-packages'] |
| | platform | En qué tipo de sistema operativo está corriendo Python | sys.platform | "win32" |

# Sistema de librerias

| Tipo | Nombre | Propósito | Ejemplo | Resultado |
|---|---|---|---|---|
| | stdin | Entrada estándar | sys.stdin.readline() | (Típicamente) la siguiente línea de entrada desde el teclado |
| | stdout | Salida estándar | sys.stdout.write('****') | (Típicamente) imprime cuatro asteriscos en la pantalla |
| | stderr | Error estándar | sys.stderr.write ('Program crashing!\n') | Imprime un mensaje de error en la pantalla |
| | version | Qué versión de Python es la que se está utilizando | sys.version | "2.4 (#60, Feb 9 2005, 19:03:27) [MSC v.1310 32 bit (Intel)]" |
| Función | exit | Sale de Python, regresando un código de status al sistema operativo | sys.exit(0) | Termina el programa con status 0 |
| | | | | |

# Argumentos de linea de comandos

- sys.argv contiene los argumentos de línea de comandos del programa
  - El nombre del programa siempre es sys.argv[0]

```
import sys

for i in range(len(sys.argv)):
    print i, sys.argv[i]

$ python command_line.py

0 command_line.py

$ python command_line.py first second

0 command_line.py
1 first
2 second
```

# Entrada/Salida Estándar

- sys.stdin y sys.stdout son la entrada y salida estándar
  - Normalmente conectadas al teclado y a la pantalla
  - Si se redirecciona o se utiliza una tubería, el sistema oprativo las conecta a archivos u otros programas
- sys.stderr está ligado al error estándar

# Entrada/Salida Estándar

```
import sys

count = 0
for line in sys.stdin.readlines():
    count += 1

sys.stdout.write('read ' + str(count) + ' lines')

$ python standard_io.py < standard_io.py

$ read 7 lines
```

# Ruta de búsqueda

- 
- sys.path es el listado de lugares donde Python es capaz de buscar a fin de encontrar módulos a importar
  - Inicializado a partir de la variable de ambiente PYTHONPATH (environment variable)
  - El directorio que contiene el programa que se ejecuta actualmente se posiciona al inicio de la lista
- Si sys.path is ['/home/swc/lib', '/Python24/lib'], entonces import geology ejecutará:
  - ./geology.py
  - /home/swc/lib/geology.py
  - /Python24/lib/geology.py
  - Después falla

# Salida

- sys.exit termina el programa
- Regresa al sistema operativo un código de status (status code) entero
  - 0 indica ejecución exitosa ("cero errores")
  - Diferente de cero es un código de error
  - Sí, es lo opuesto a lo que se esperaría...
- De no salir explícitamente, Python regresa 0
  - Así que se recomienda utilizar sys.exit(1) o algo similar para que el sistema operativo detecte si ocurrió un error

# Librería math

- Mucho de la librería estándar de Python es un envoltorio de las librerías standard de C
  - Veáse el resumen (Summary) de cómo envolver librerías (encapsular) por uno mismo
- Ejemplo: la librería math

# Librería math

| Tipo | Nombre | Propósito | Ejemplo | Resultado |
|---|---|---|---|---|
| Constante | e | Constante | e | 2.71828... |
| | pi | Constante | pi | 3.14159... |
| Función | ceil | Ceiling | ceil(2.5) | 3.0 |
| | floor | Floor | floor(-2.5) | -3.0 |
| | exp | Exponencial | exp(1.0) | 2.71828... |
| | log | Logaritmo | log(4.0) | 1.38629... |

# Trabajando con el sistema de archivos

- El módulo os es una interface entre Python y el sistema operativo
  - Intenta ocultar las diferencias entre distintos sistemas operativos
  - Pero puede hacer muchas cosas

# Trabajando con el sistema de archivos

| Tipo | Nombre | Propósito | Ejemplo | Resultado |
|---|---|---|---|---|
| Constante | curdir | Nombre simbólico de directorio actual (current directory) | os.curdir | . on Linux or Windows. |
| | pardir | Nombre simbólico de directorio padre (parent directory) | os.pardir | .. en Linux o Windows. |
| | sep | El caracter separador utilizado en rutas | os.sep | / en Linux, \ en Windows. |
| | linesep | Marcador de final de línea utilizado en archivos de texto | os.linesep | \n en Linux, \r\n en Windows. |
| Función | listdir | Lista el contenido de un directorio | os.listdir('/tmp') | Los nombres de todos archivos y directorios en /tmp (excepto . y ..). |

# Trabajando con el sistema de archivos

| Tipo | Nombre | Propósito | Ejemplo | Resultado |
|---|---|---|---|---|
| | mkdir | Crea un nuevo directorio | os.mkdir('/tmp/scratch') | Crea el directorio /tmp/scratch. Utiliza os.makedirs para crear varios directorios en una sola instrucción. |
| | remove | Borra un archivo | os.remove('/tmp/workingfile.txt') | Borra el archivo /tmp/workingfile.txt. |
| | rename | Renombra (o mueve) un archivo o rirectorio. | os.rename('/tmp/scratch.txt', '/home/swc/data/important.txt') | Mueve el archivo /tmp/scratch.txt a /home/swc/data/important.txt. |
| | rmdir | Remueve un directorio. | os.rmdir('/home/swc') | Probablemente no es algo que se desee hacer. Utiliza os.removedirs para remover varios directorios en una sola instrucción. |
| | stat | Obtiene la información de un directorio. | os.stat('/home/swc/data/important.txt') | Obtiene la fecha de creación, el tamaño, etc., de important.txt |
| | | | | |

# Trabajando con el Sistema de Archivos

```
import sys, os

print 'initial working directory:', os.getcwd()
os.chdir(sys.argv[1])
print 'moved to:', os.getcwd()
print 'contents:', os.listdir(os.curdir)
```

```
$ python os_example.py ~/swc
```

```
initial working directory: /home/dmalfoy/swc/lec/inc/py03
moved to: /home/dmalfoy/swc
contents: ['.svn', 'conf', 'config.mk', 'data', 'depend.mk', 'thesis']
```

# Estado de un archivo o directorio

- os.stat regresa un objeto cuyos elementos poseen información acerca de un archivo o directorio, incluyendo:
  - st_size: tamaño en bytes
  - st_atime: fecha del acceso más reciente
  - st_mtime: fecha de la modificación más reciente

# Estado de un archivo o directorio

```
import sys
import os

for filename in sys.argv[1:]:
    status = os.stat(filename)
    print filename, status.st_size, status.st_atime

$ python stat_file.py . stat_file.py

. 0 1137971715
stat_file.py 141 1137971715
```

# Estado de un archivo o directorio

- Los fechas se miden en segundos desde el epoch
  - En Unix, el epoch is medianoche, Enero 1, 1970
  - Revisión rápida: 1137971715 / (60 * 60 * 24 * 365) es 36 años
- Utiliza el módulo time para convertir los tiempos a formato entendible para los humanos
  - Es más complicado de lo que se imagina...

# Manipulación del Path name

- 
- os posee un submódulo llamado os.path
  - Manipula nombres de rutas correcta y eficientemente
  - No escribir funciones propias para esto---las reglas son más complicadas de lo que se imagina
- 

# Manipulación del Path name

| Tipo | Nombre | Propósito | Ejemplo | Resultado |
|---|---|---|---|---|
| Función | abspath | Crea nombres de rutas absolutos normalizados. | os.path.abspath('../jeevan/bin/script.py') | /home/jeevan/bin/script.py (si se ejecuta en /home/gvwilson) |
| | basename | Regresa la última parte de una ruta (tal como el nombre del archivo o el nombre del último directorio). | os.path.basename('/tmp/scratch/junk.data') | junk.data |
| | dirname | Regresa todo excepto la última parte de la ruta. | os.path.dirname('/tmp/scratch/junk.data') | /tmp/scratch |
| | exists | Regresa True si el nombre de una ruta se refiere a un archivo o directorio existente. | os.path.exists('./scribble.txt') | True si existe un archivo llamado scribble.txt en el directorio de trabajo actual, False de lo contrario. |
| | getatime | Obtiene la fecha del último acceso de un archivo o directorio (como os.stat). | os.path.getatime('.') | 1112109573 (lo que significa que el directorio actual fue leído o escrito a las 10:19:33 EST en Marzo 29, 2005). |

# Manipulación del Path name

| Tipo | Nombre | Propósito | Ejemplo | Resultado |
|---|---|---|---|---|
| | getmtime | Obtiene la fecha de la última modificación de un archivo o directorio (como os.stat). | os.path.getmtime('.') | 1112109502 (lo cual significa que el directorio actual fue modificado 71 segundos antes que la fecha mostrada arriba). |
| | getsize | Obtiene el tamaño de algo en bytes (como os.stat). | os.path.getsize ('py03.swc') | 29662. |
| | isabs | True si su argumento es un nombre de ruta absoluto. | os.path.isabs('tmp/ data.txt') | False |
| | isfile | True si su argumento identifica un archivo existente. | os.path.isfile('tmp/ data.txt') | True si un archivo llamado ./tmp/data.txt existe, y False de lo contrario. |
| | isdir | True si su argumento identifica un directorio existente. | os.path.isdir('tmp') | True si el directorio actual contiene un subdirectorio llamado tmp. |
| | join | Une fragmentos del nombre de ruta para crear un nombre de ruta completo. | os.path.join('/tmp', 'scratch', 'data.txt') | "/tmp/scratch/data.txt" |

# Manipulación del Path name

| Tipo | Nombre | Propósito | Ejemplo | Resultado |
|---|---|---|---|---|
| | normpath | Normaliza un nombre de ruta (como ejemplo remueve diagonales redundantes, utiliza . y .., etc.). | os.path.normpath('tmp/scratch/../other/file.txt') | "tmp/other/file.txt" |
| | split | Regresa ambos valores regresados por os.path.dirname y os.path.basename. | os.path.split('/tmp/scratch.dat') | ('/tmp', 'scratch.dat') |
| | splitext | Divide una ruta en dos partes root y ext, de tal forma que ext sea la última pieza que inicie con un ".". | os.path.splitext('/tmp/scratch.dat') | ('/tmp/scratch', '.dat') |

# Sumario

- La verdadera forma de medir un lenguaje de programación es qué tan bien soporta la modularización
- Utilice funciones, librerías, y Programación Básica Orientada a Objetos en Python para conservar los programas comprehensibles
  - Recuerde que los está escribiendo para otros seres humanos