



-Minino de Cheshire, ¿podrías decirme, por favor, qué camino debo seguir para salir de aquí?

-Esto depende en gran parte del sitio al que quieras llegar -dijo el Gato.

-No me importa mucho el sitio... -dijo Alicia.

-Entonces tampoco importa mucho el camino que tomes - dijo el Gato.

- ... siempre que llegue a alguna parte - añadió Alicia como explicación.

- ¡Oh, siempre llegarás a alguna parte - aseguró el Gato -, si caminas lo bastante!

Programación Funcional

PYTHON

Capitán Luis Tomás Wayar

Programación funcional

- Es un paradigma de programación que está basado en funciones.
- Python no es un lenguaje funcional puro, sin embargo incorpora esta característica.
- Se entiende el concepto de función según su definición matemática y no como simples subprogramas de los lenguajes imperativos.

“Lo que yo iba a decir - siguió el Dodo en tono ofendido - es que el mejor modo para secarnos sería una Carrera Loca.



Funciones de orden superior

- Son funciones que reciben o devuelven funciones como parámetros.
- Operan sobre funciones, es decir el dominio o la imagen de estas funciones son funciones.

```
def superior(interna):
    def funcion1():
        print "Funcion 1"

    def funcion2():
        print "Funcion 2"
    funcion = {"uno": funcion1,
               "dos": funcion2}
    return funcion[interna]

nuevafuncion = superior('dos')
nuevafuncion()
```

```
def funcion1():
    print "Funcion 1"

def funcion2():
    print "Funcion 2"

def superior(funcion):
    funcion()

superior(funcion1)
superior(funcion2)
```



“Si conocieras al Tiempo tan bien como lo conozco yo - dijo el Sombrerero -, no hablarías de matarlo. ¡El Tiempo es todo un personaje!”

Funciones Lambda

Son funciones anónimas, mínimas definidas en una línea, no pueden contener **bucles** y no pueden utilizar la palabra clave **return** para regresar un valor.

lambda <parámetros>:<expresión></expresión></parámetros>

```
cuadrado = lambda x: x ** 2
print cuadrado(2)

Lista = range(10)

for numero in Lista:
    print cuadrado(numero)
```

“- ¿He perdido la razón?

- Me temo que sí. Estás demente. Pero te diré un secreto. Las mejores personas lo están.”



Funciones de alto orden (map)

map: aplica una función sobre cada ítem de una lista y devuelve una lista donde se le aplicó la función a cada elemento

`nueva_lista = map(función, lista)`

```
lista = [0, 1, 2, 4, 5, 6, 7, 8, 9]

Lista_nueva = map(lambda x: x ** 2, lista)

print Lista_nueva
```

“Sí yo hiciera mi mundo todo sería un disparate. Porque todo sería lo que no es. Y entonces al revés, lo que es, no sería y lo que no podría ser si sería. ¿Entiendes?”



Funciones de alto orden (reduce)

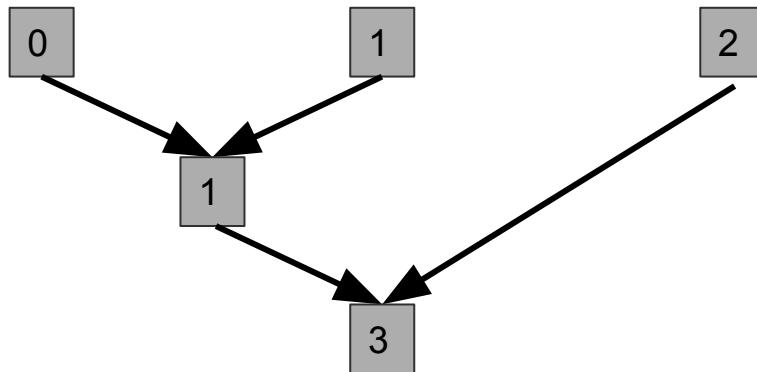
reduce: reduce los ítems de una lista aplicando una función reductora, devuelve un solo valor

valor = reduce(función, lista, (primero))

```
Lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

sumatoria = reduce(Lambda x, y: x + y, Lista)

print sumatoria
```



*- ¿Cuánto es para siempre?
- A veces, solo un segundo.*



Funciones de alto orden (filter)

filter: coteja que los elementos de una lista cumplan con cierta condición expresada en una función boolean.

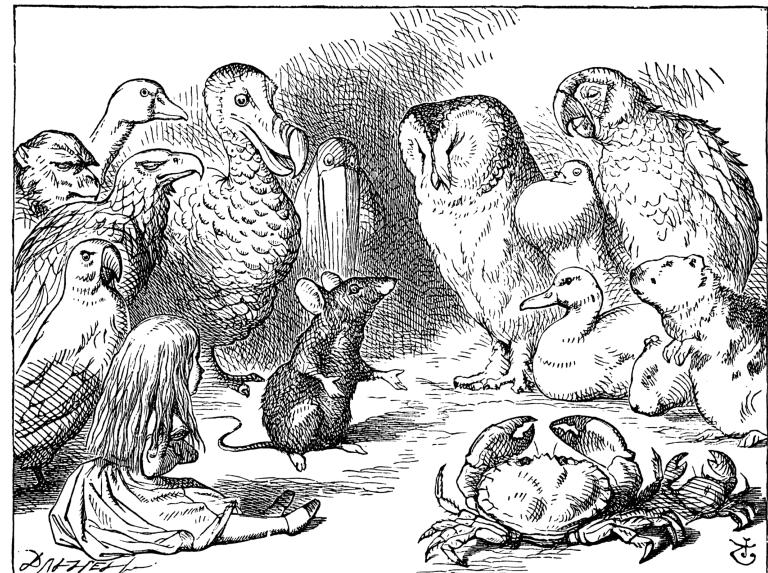
`nueva_lista = filter(funcion, lista)`

```
Lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

pares = filter(lambda x: x % 2 == 0, Lista)

print pares
```

¡Qué extraño es todo hoy! ¡Y ayer sucedía todo como siempre!... ¿Habré cambiado durante la noche? Pero si no soy la misma, el asunto siguiente es ¿quién soy? ¡Ay, ése es el gran misterio!



Funciones de alto orden (zip)

zip: función para reorganizar listas, admite varias listas, toma cada elemento iésimo de cada lista y los une en una tupla, luego hace una lista con ellas

lista = zip(lista, lista)

```
animal = ["conejo", "gato", "liebre", "oruga", "pajaro"]
nombre = ["blanco", "de cheshire", "de marzo", "azul", "dodo"]

personajes = zip(animal, nombre)

print personajes
```

"Desde que me caí por esa madriguera me han dicho qué tengo hacer y quién debo ser. Me han encogido, aumentado, arañado y metido en una tetera, me han acusado de ser Alicia y de no ser Alicia, pero éste es mi sueño, y yo decidiré cómo continúa"



Listas por comprensión

Es una construcción compacta que permite mapear y/o filtrar una lista en otra aplicando una función a cada elemento de la lista, opcionalmente la lista original puede ser filtrada

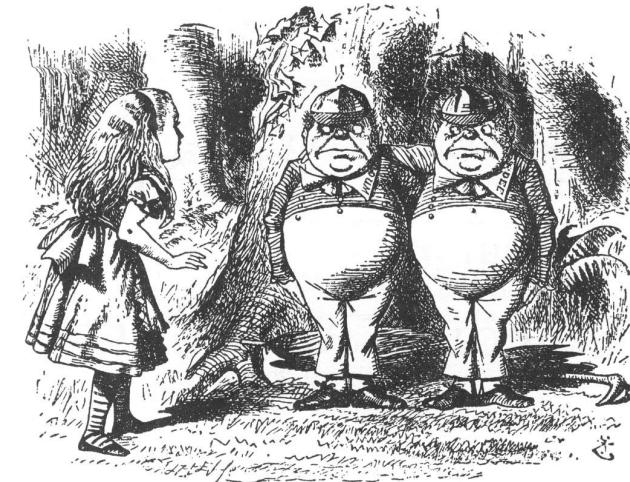
Crear una lista con los cuadrados de los dígitos:

```
print [digito**2 for digito in range(10)]
```

Crear una lista con los cuadrados de los dígitos pares

```
print [digito**2 for digito in range(10) if digito % 2 == 0]
```

Alicia no tenía la menor idea de lo que era la latitud, ni tampoco la longitud, pero le pareció bien decir estas palabras tan bonitas e impresionantes.



Generadores

Son una clase especial de funciones que no generan una lista, sino una secuencia de valores sobre el que iterar

Pueden ser:

- **Métodos generadores:** incluyen la palabra “**yield**” y se declaran como funciones
- **Expresión generadora:** se declara como listas por comprensión entre “**()**”

```
def generador_ip(cantidad):
    for i in range(cantidad):
        a = random.randint(0,255)
        b = random.randint(0,255)
        c = random.randint(0,255)
        d = random.randint(0,255)
        yield str(a)+"."+str(b)+"."+str(c)+"."+str(d)

for i in generador_ip(5):
    print i
```

```
gen_pares = (p for p in range(10) if p % 2 == 0)

print gen_pares.next()
print gen_pares.next()
```

“Alicia se daba por lo general muy buenos consejos a sí misma (aunque rara vez los seguía).”



Decoradores

- Son funciones que reciben una función como parámetro y devuelve otra función.
- Cambian el comportamiento de un función, método o clase sin modificar su código.
- Crea un “envoltura” alrededor de la función a decorar, esta envoltura y la función original son la nueva función que obtenemos.

```
def decorador(funcion):
    print "Decoracion de"
    return funcion

@decorador
def funcionPrueba():
    print "una funcion"

funcionPrueba()
```

De modo que ella, sentada con los ojos cerrados, casi se creía en el país de las maravillas, aunque sabía que sólo tenía que abrirlos para que todo se transformara en obtusa realidad.



FIN

Muchas gracias

“Es una pobre clase de memoria que sólo funciona hacia atrás.”

“Y cuando termines de hablar... ¡te callas!”

