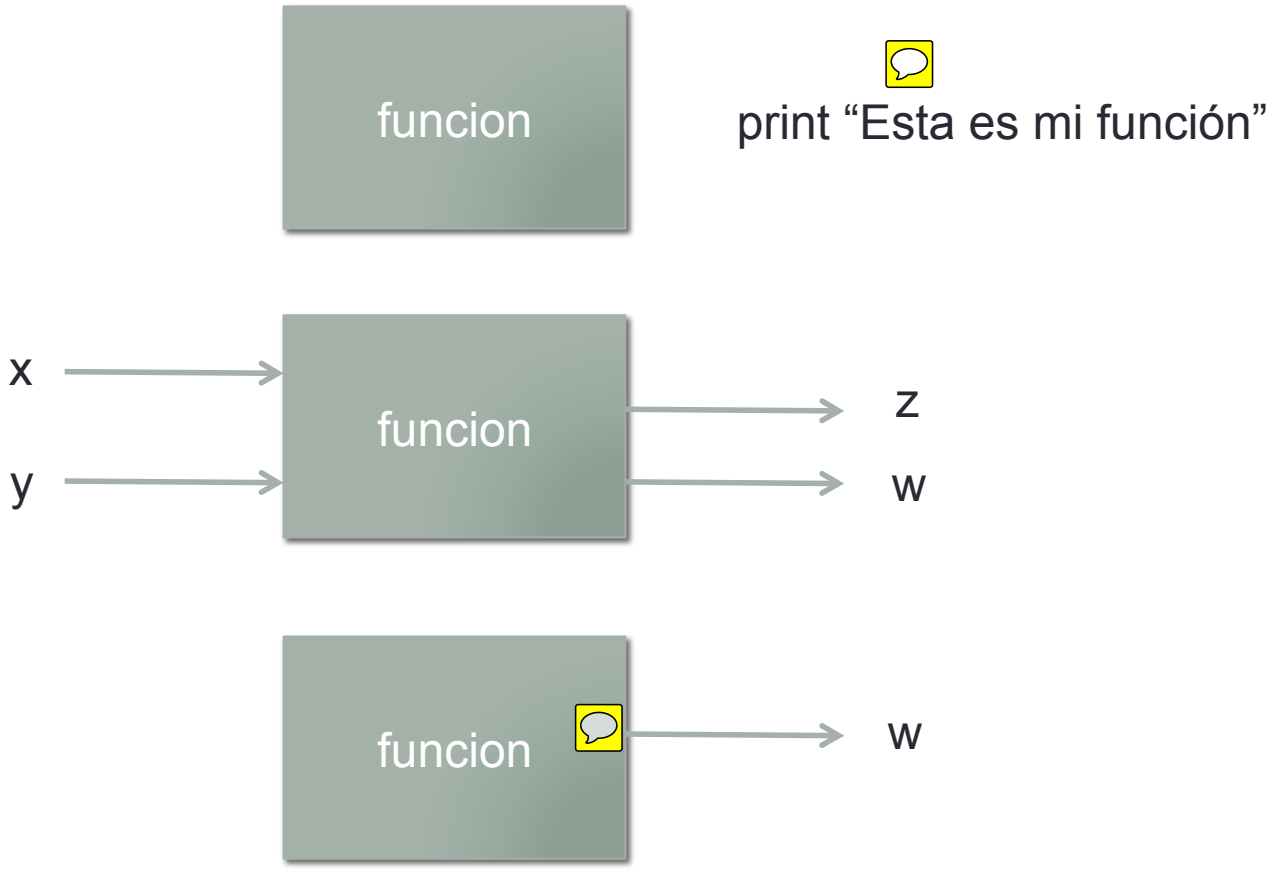
PROGRAMACIÓN EN PYTHON 2

Clara Higuera

Laboratorio Integrado de Biofísica y Bioinformática

Nov-2015

Funciones



Funciones

- Las funciones sirven para encapasular código y pueden tener o no tener parámetros de entrada
  - Definición:

```
def nombre_funcion():
    print "Hola Mundo"
```

  - Llamada:
    - Una función, no es ejecutada hasta que no sea llamada. Para llamar a una función, simplemente se indica su nombre.

```
funcion()
```

Funciones

- Cuando una función queremos que devuelva un dato utilizamos *return*. Este valor puede ser utilizado para asignarlo a una variable.

```
def suma(a,b):
    return a+b

resultado= suma(3,4)
print resultado
```

```
def fun_nombre(cadena1,cadena2):
    mi_nombre=cadena1+" "+cadena2
    return mi_nombre

nombre1="Jose"
apellido1="Martinez"

nombre_completo=fun_nombre(nombre1,apellido1)
print nombre_completo
```

Funciones

- Cuando una función queremos que devuelva un dato utilizamos *return*. Este valor puede ser utilizado para asignarlo a una variable.

```
def suma(a,b):
    return a+b

resultado= suma(3,4)
print resultado
```

```
def fun_nombre(cadena1,cadena2):
    mi_nombre=cadena1+" "+cadena2
    return mi_nombre

nombre1="Jose"
apellido1="Martinez"

nombre_completo=fun_nombre(nombre1,apellido1)
print nombre_completo
```

Podemos llamar a una función pasándole literales (numeros, cadenas,…) o variables

Funciones

- Podemos utilizar cualquier tipo de datos de entrada y salida
- Para documentar la función usamos con comillas triples debajo de la definición

Devuelve 2 elementos

```
#Definición de la función
def info_lista(lista):
    """Esta funcion devuelve el primer y ultimo elemento
    de una lista"""
    return lista[0],lista[len(lista)-1]

#LLamada
l=[25,45,23,5,65,54,4]
primero,ultimo = info_lista(l)
print"El primer elemento es :",primero,"\n"
print"El ultimo elemento es :",ultimo,"\n"
```

Funciones

- Ejercicio 1
  - Definir una función que reciba como parámetros dos variables numéricas y devuelva la media
  - Llamar a la función e imprimir el resultado
- Ejercicio 2
  - Definir una función que reciba como parámetros 2 cadenas y devuelva una cadena resultado de la concatenación de las dos cadenas

Modulos

- Forma de **reutilizar** código
- Agrupan funciones y objetos relacionados
- Por ejemplo, **math**
- Hay que importarlos

1. Todo el módulo

```
>>import math

>>> math.sqrt(2)
1.4142135623730951
```

2. Todo el módulo, con un alias, para abreviar

```
>>> import math as m

>>> m.sqrt(2)
1.4142135623730951
```

3. Importar dentro del espacio de nombres actual

```
>>> from math import sqrt

>>> sqrt(2)
1.4142135623730951
```

Cuidado. Podemos reemplazar funciones preexistentes

Modulos

- Buscar ayuda:

```
>>> import math
>>> help(math)      # Ayuda del modulo
>>> help(math.sqrt) # Ayuda de la función
```

Ejercicio:

- Usando la ayuda (o internet)
  - Buscar como hacer senos, cosenos y exponenciales
  - Logaritmos neperianos y decimales
  - Redondeos al entero superior, al inferior y al más cercano
  - Valores absolutos

Modulos: numpy

- Las listas se pueden utilizar para simular vectores y matrices, pero son bastante inconvenientes
- Por ejemplo, la suma de dos listas no suma los componentes (no tendría sentido: distintos tipos)

- El **módulo numpy** proporciona arrays eficientes

```
>>> import numpy as np
>>> vector = np.array([1,2,3])
```

- Datos de un solo tipo (todos float, todos int)
- Acceso como en listas y tuplas

```
>>> vector[0]
>>> vector[2:5]
```

Modulos: numpy

- También podemos crear matrices

```
>>> matriz = np.array([[1,2,3],[0,1,0]])
```

- Acceso

```
>>> matriz[0,2]
>>> matriz[0][2]
3
```

- Filas y columnas:

```
>>> matriz[0]; matriz[0,:]
>>> matriz[:,0]
```

Devuelven la primera fila

Devuelven la primera columna

Modulos: numpy

- Operaciones basicas

```
>>> v = np.array([1,2,3])
>>> u = np.array([3,2,1])
>>> v + u
array([4, 4, 4])
```

- Método para obtener el número de dimensiones:

```
>>> matriz.ndim
2
```

- Forma de la matriz: (num filas, num columnas)

```
>>> matriz.shape
(2, 3)
```

- Siempre que se mantenga el número de elementos, la forma se puede alterar

```
>>> matriz.shape = (3,2)
```

Modulos: numpy

- **Funciones útiles**

  Vector de números enteros:

  - np.arange(10)
  - np.arange(6,12,2)

  (inicio,fin, incremento)

  Se excluye el último!
  Podemos poner, 1, 2 o 3 args

  Rango de valores reales

  - np.linspace(0,1,10)

  array([0. , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
  0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.])

  Matriz de unos:

  np.ones((3,3))

  - Matriz diagonal:
    - np.diag([1,2,3])

Modulos: numpy

- **Ejercicio 3:**
  - Buscar en la ayuda (o internet) como hacer la matriz identidad y cómo generar números aleatorios

- **Ejercicio 4:**

- Crear los siguientes arrays
  - Vector con números descendentes del 30 al 0, de 3 en 3
  - Matriz identidad 5x5
  - Matriz con números del 1 al 5 en la diagonal
  - Lo mismo pero del 2 al 10, de 2 en 2
  - ¿Qué submódulo permite generar números aleatorios?
  - Hacer una matriz de 2x3 de números aleatorios

Modulos: numpy

- **EJERCICIO PARA EL CAMPUS:**
  - Escribir una función que tome una matriz y muestre uno a uno sus elementos, indicando para cada uno su fila y columna.
  - Pista: utilizar el atributo de matrices .shape

Más sobre listas, cadenas y diccionarios

- El operador **in** permite averiguar si un elemento pertenece a una lista o cadena

```
>>> nombre = "Tarquinio"
>>> lista = ['rojo','verde',35]
>>> "a" in nombre
True
>>> "qui" in nombre
True
>>> "e" in nombre
False
>>> 35 in lista
True
```

- Devuelve booleano: se puede usar en condiciones de **if** y **while**

Más sobre listas, cadenas y diccionarios

- En diccionarios, **in** comprueba claves, no valores

```
>>> dicc = {"A": 30, "B": "C"}
>>> "A" in dicc
True
>>> "C" in dicc
False
```

- Para comprobar si está un valor:

```
>>> "C" in dicc.values()
True
```

- Para evitar que una busqueda por clave de error, usar **get**

```
>>> dicc["D"] # ERROR
>>> dicc.get("D","No existe")
'No existe'
```

Más sobre listas, cadenas y diccionarios

En diccionarios y listas (pero no en tuplas ni cadenas) se pueden eliminar elementos usando **del**

```
>>> dicc = {"A": 30, "B": "C"}
>>> del dicc["A"]
>>> dicc
{"B": "C"}
>>> lista = [0,10,20,30,40]
>>> del lista[1]
>>> [0, 20, 30, 40]
```

- En listas también funcionan rangos

```
>>> del lista[3:5]
```

- Y, también en listas, se puede eliminar por valor

```
>>> lista.remove(30)  # Elimina el primer elemento 30
```

Método asociado al objeto lista

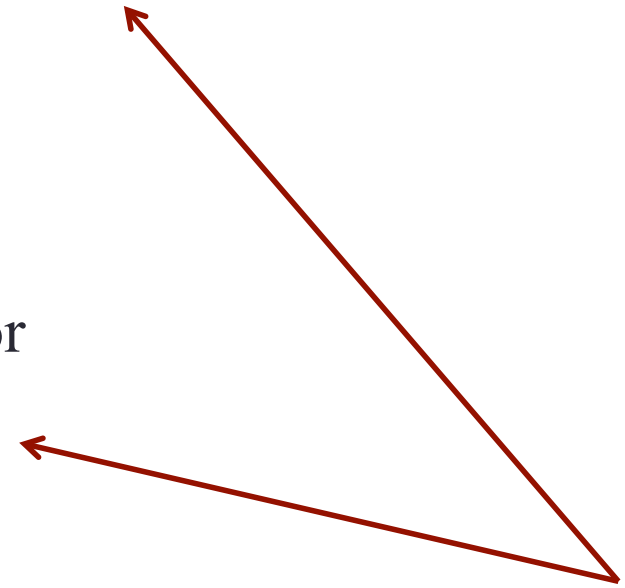
Más sobre listas, cadenas y diccionarios

- En diccionarios, asignar una nueva clave equivale a insertarla

```
>>> mapa = dict([("A",30),("B",10)]) # Lista de tuplas
>>> mapa["C"] = 15
>>> mapa
{'A': 30, 'C': 15, 'B': 10}
```

- Si la clave existía, se sustituye el valor

```
>>> mapa = dict(A = 30, B = 10)
>>> mapa
{'A': 30, 'B': 10}
>>> mapa["A"] = 2
>>> mapa
{'A': 2, 'B': 10}
```



Formas alternativas de crear diccionarios, usando el constructor `dict`

Más sobre listas, cadenas y diccionarios

- En listas, las inserciones se hacen con **insert**

```
>>> timeline = [1666,1789,1863,1945]
>>> timeline.insert(2,1812)
>>> timeline
[1666, 1789, 1812, 1863, 1945]
```

- También con **append**, si son al final (o si es vacía)

```
>>> timeline.append(1989)
>>> timeline[1666, 1789, 1812, 1863, 1945, 1989]
>>> nombres = []
>>> nombres.append('Manolo')
>>> nombres
>>> ['Manolo']
```

Más sobre listas, cadenas y diccionarios

- El último elemento de una lista se puede obtener con **pop**

```
>>> proteinas = ['Ras','PLC','EGFR']
>>> proteinas.pop()
'EGFR'
>>> proteinas
['Ras', 'PLC']
```

- Admite índices concretos

```
>>> lista = ['A','B','C','D','E']
>>> d = lista.pop(3)
>>> lista
['A','B','C','E']
```

Más sobre listas, cadenas y diccionarios

- Las listas se pueden ordenar con **sort**

```
>>> cuentas = [1,34,12,7,21]
>>> cuentas.sort()
>>> cuentas
[1, 7, 12, 21, 34]
```

- Y se pueden invertir con **reverse**

```
>>> lista = ['A','B','C','D','E']
>>> lista.reverse()
>>> lista
['E', 'D', 'C', 'B', 'A']
```

Más sobre listas, cadenas y diccionarios

Ejercicio 5.

Escribir un script que haga lo siguiente:

1. Cree una lista que contenga los nombres de las bases: Adenina, Guanina, Citosina, Timina
2. Añadir una más al final
3. Ordenar la lista alfabéticamente
4. Eliminar la base en la segunda posición
5. Invierte la lista
6. Imprimir el número de elementos

Imprimir la lista después de cada paso.

Más sobre listas, cadenas y diccionarios

Por desgracia, reverse no funciona con cadenas. ¿Qué podemos hacer?

Opción 1:

- Se puede convertir una cadena en una lista

```
>>> deletreo = list("hola")
>>> deletreo
['h', 'o', 'l', 'a']
>>> deletreo.reverse()
['a', 'l', 'o', 'h']
```

- Para devolver la cadena, usar **join**

```
>>> "".join(deletreo)
'aloh'
```

- Las comillas encierran un separador. Por ejemplo:

```
>>> "-".join(deletreo)
'a-l-o-h'
```

Más sobre listas, cadenas y diccionarios

Por desgracia, reverse no funciona con cadenas. ¿Qué podemos hacer?

Opción 1:

- Se puede convertir una cadena en una lista

```
>>> deletreo = list("hola")
>>> deletreo
['h', 'o', 'l', 'a']
>>> deletreo.reverse()
['a', 'l', 'o', 'h']
```

- Para devolver la cadena, usar **join**

```
>>> "".join(deletreo)
'aloh'
```

- Las comillas encierran un separador. Por ejemplo:

```
>>> "-".join(deletreo)
'a-l-o-h'
```

Opción 2:

Cadena="hola"
Cadena=Cadena[::-1]

Más sobre listas, cadenas y diccionarios

- Más operaciones con cadenas

```
>>> dna = 'aggtctagtacgtagctaggtaacgtat\n'
```

- Cambio entre mayúscula y minúsculas

```
>>> dna.upper() # A mayusculas
>>> dna.lower() # A minúsculas
```

- Eliminar caracteres blancos (espacios, '\n', '\t') en los extremos

```
>>> dna.rstrip() # Elimina a la derecha
>>> dna.lstrip() # Elimina a la izquierda
>>> dna.strip()  # A ambos lados
>>> "AGCCGG".rstrip("G") # Elimina las Gs a la derecha
'AGCC'
```

COMPROBAR SIEMPRE CÓMO FUNCIONA LA FUNCIÓN QUE VAIS A USAR, UNAS MODIFICAN EL OBJETO Y OTRAS DEVUELVEN UN OBJETO NUEVO

Más sobre listas, cadenas y diccionarios

- Más operaciones con cadenas

```
>>> dna = 'aggtctagtacgtagctaggtacgtat\n'
```

- Comienzo y fin

```
>>> dna.startswith('aggt') # True
>>> dna.endswith('cg') # False
>>> dna.startswith('ct',4) # A partir de dna[4]
True
```

- Busquedas (ver también expresiones regulares)

```
>>> dna.find('tacg') # Primera aparicion
8
>>> dna.find('tacg',9) # Desde la posicion 9
20
>>> dna.rfind('tacg') # Primera por la derecha
20
```

Más sobre listas, cadenas y diccionarios

- Más operaciones con cadenas

```
>>> dna = 'aggtctagtacgtagctaggtacgtat\n'
```

- Sustituciones

```
>>> dna.replace('tacg', 'TACG')
'aggtctagTACGtagctaggTACGtat\n'
>>> dna.replace('g', 'G', 3) # Solo las 3 primeras
'aGGtctaGtacgtagctaggtacgtat\n'
```

- Generar lista mediante separador

```
# Normalmente se usa un solo caracter
# (o ninguno para separa por espacio blanco)
>>> dna.split("ct")
['aggt', 'agtacgtag', 'aggtacgtat\n']

# Lo opuesto (join) ya lo hemos visto
```

Más sobre listas, cadenas y diccionarios

- Algunas operaciones con cadenas

```
>>> dna = 'aggtctagtacgtagctaggtacgtat\n'
```

- Cuenta (apariciones no solapantes)

```
>>> dna.count('ct')
2
```

Ejercicio 6:

Dada la cadena:

DNA="ACGTGTGACGCATGCGTGAGTATGAGTG"

- Reemplazar la timina por el uracilo
- Si la subcadena “GUG” aparece en la cadena imprimir un mensaje que indique la primera posición en la que aparece y cuantas veces.

Ejercicio 7:

Dada la cadena dna = ' aggtctagtacg\ntagctaggtacgtat\n'

- Eliminar todos los caracteres blancos
- Invertir el orden de la secuencia y pasar a mayúsculas
- Hacer una lista con todos los tripletes (solapantes) y ordenarla alfabéticamente