INTRODUCCIÓN A LA PROGRAMACIÓN EN PYTHON

Clara Higuera

Laboratorio Integrado de Biofísica y Bioinformática

Nov-2015

Conceptos básicos

- ¿Qué es un programa?
- Desarrollo y ejecución de un programa
- Datos y operaciones sobre datos
- Control de flujo de un programa 
  - Sentencias de control
- Entrada/Salida de un programa
- Funciones
- Distintos lenguajes de programación: componentes básicos comunes

Conceptos básicos

- Tipos de lenguajes:
  - **Nivel del lenguaje**
    - Bajo nivel
      - Código máquina (ceros y unos)
      - Ensamblador
    - Alto nivel: Se parece más al lenguaje humano
      - Fortran
      - C
      - Java
      - Python
      - Perl

Rápidos, optimizados al hw de la máquina
Pero poco portables

Lenguajes más expresivos
Menos óptimizados, más lentos

Conceptos básicos

- Tipos de lenguajes
  - **Tipo de ejecución**
    - Lenguaje compilado: *Antes de ejecutar se traduce a código máquina*
      - *Rápido, muy optimizado para cada tipo de máquina*
      - *La programación es más laboriosa*
      - C, Fortran, Java
    - Lenguaje Interpretado
      - *Un intermediario (interprete de comandos) ejecuta una a una las instrucciones que vamos dando.*
      - *Más lento: cada instrucción se debe interpretar en tiempo real*
      - *Fácil de programar: los cambios se pueden comprobar inmediatamente*
      - *Python, Perl, Matlab*

Conceptos básicos

- Todo lenguaje de programación viene descrito por:
- Un léxico
  - Palabras clave
  - Nombres de variables, constantes, funciones predefinidas..
- Una sintaxis
  - Reglas con las que deben combinarse los elementos de un lenguaje de programación (léxico/términos y puntuación) para construir frases/instrucciones
- Una semántica
  - El significado o propósito del programa sintácticamente correcto.

Introducción

- Qué es Python
- Es un lenguaje interpretado de alto nivel 
- Creado por Guido van Rossum en 1989
- Tiene una sintaxis sencilla, con estructuras de datos y de control muy potentes
- Fácil de aprender, de leer y de mantener
- Open Source
- Gran cantidad de módulos muy útiles
  - math : funciones matemáticas básicas
  - numpy : cálculo numérico
  - scipy : cálculo científico
  - pylab/matplotlib : representaciones gráficas

Introducción

- Tutorial oficial
  - https://docs.python.org/2/tutorial/
- Cursos on line
  - https://www.codecademy.com/es/learn/python
  - http://www.learnpython.org/
- Anaconda
  - iPython: Intérprete de comandos
  - Spider: Entorno de desarrollo
  - https://www.continuum.io/downloads
  - Versión Python 2.7

Python: intérprete de comandos

- Abrimos Anaconda
  - Seleccionamos Spider: Entorno de desarrollo
  - Utilizamos la consola como una calculadora interactiva

>>2+2 → Igual que una calculadora, es interactivo

>>_+2 → Recuerda el último resultado (útil en el modo interactivo)

  - Repetimos lo mismo en el editor y ejecutamos
    2+2
  - Para obtener resultado en consola:
    print 2+2

Python: intérprete de comandos

- Multiples operaciones numéricas básicas
  - Suma: +
  - Resta: -
  - División: /
  - División entera: //
  - Resto (modulo): %
  - Potencia: **

¡CUIDADO!

```
>>> 3 ** 3
27
>>> 3 ^ 3
0
```

XOR

Operación bit a bit

```
>>> 3. / 2. * 4.
6.0
```

Para evitar ambigüedades, usad paréntesis en expresiones complejas (¡o cuando sea imprescindible!)

```
>>> 3. / (2. * 4.)
0.375
```

Literales, variables y operadores

- Literales: Mantienen siempre su valor
  - Números
  - Cadenas de texto
  - Booleanos
- Variables: Pueden variar a lo largo del programa
- Operadores: Manipulan variables y literales
  - Operadores de números
  - Operadores de cadenas
  - Operadores lógicos
  - Operadores de comparación

Tipos de datos

- En python los tipos no se especifican son implicitos
- Tipos principales:
  - Enteros (int): 1, 2, 3 ,4 ..
  - Reales (float): 1.3, 1.5, 489.9, 1.
  - Booleanos (bool): True, False
  - Cadenas de texto (string): “Hello”, ‘Hello’
- La funcion type() nos devuelve el tipo de un elemento

```
>>> type(676)
<type 'int'>

>>> type(True)
<type 'bool'>

>>> type(56)
<type 'int'>
```

Variables

- Variables
  - Para poder manejar los datos es necesario almacenarlos en variables. Sus valores pueden variar a lo largo del programa

```
>>> a = 3; b = 2
>>> a + 4
7
>>> a + 1.0 # Conversión a float
4.0
>>> a + b
5
```

Cadenas

- Hay varias formas de definirlas:

```
>>> str1 = "Hola mundo"
>>> str2 = 'Hola mundo'
```

- Permiten incluir comillas

```
>>> name = "O'connor"
>>> quote = 'Diego dijo "Hola mundo"'
```

- Y cadenas con salto de linea: \n

```
>>'Donde dije digo\ndigo Diego'
```

Cadenas

- Operaciones con cadenas:
  - Repetición de n veces una cadena: str1*n
  - Concatenación de dos cadenas: str1 + str2

```
>>> "jamon"*3 # Repetición
'jamonjamonjamon'
>>> "jamon" + " serrano" # Concatenación
'jamon serrano'
```

¿Qué ocurre si mezclo tipos?

```
>>> "jamon" + 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects

>>> "jamon" + str(3) # Conversión explícita
'jamon3'
```

Cadenas

- La funcion print
  - Usamos print para imprimir el valor de una variable o un un texto descriptivo.

```
cad1= "Juan tiene 20 años"
print cad1
```

Resultado:
Juan tiene 20 años

```
cad1="Juan tiene 20 años"
cad2="Pedro tiene 22"
print cad1,"\n",cad2
```

Resultado:
Juan tiene 20 años
Pedro tiene 22

```
Disciplina1="Filosofia"
Disciplina2="Bioquímica"

print "Hablemos primero de",Disciplina1,"\n"
print "Y luego de",Disciplina2
```

Resultado:
Hablemos primero de Filosofia
Y luego de Bioquímica

Cadenas

Ejercicio:

En el editor vais a escribir vuestro primer script que realice lo siguiente:

1. Cree dos variables cadena llamadas ADN1 y ADN2 en las que se almacenen dos secuencias de ADN.
2. Imprima las dos cadenas separadas por un espacio de línea.
3. Unir las dos cadenas en una única y almacenar el resultado en una nueva variable cadena llamada DNA3
4. Imprima la cadena resultante

Cadenas

Comentario →

Imprimir por consola →

Concatenación →

```
#Creamos dos cadenas
ADN1="ACGGGAGGACGGGAAAATTACTACGGCATTAGC"
ADN2="ATAGTGCCGTGAGAGTGATGTAGTA"

#Las imprimimos por pantalla
print "La cadena 1 es:"
print ADN1+"\n"
print ADN2

#Concatenamos las dos cadenas

ADN3= ADN1+ADN2

#Impriminos el resultado
print "Esta es la cadena resultado de la concatenación"
print ADN3
```

Listas

- Tipo de datos colección: agrupar conjuntos de valores. Internamente cada posición puede ser un tipo de datos distinto.
- Se define separando sus elementos con comas y colocándolos entre [ ]
- Ejemplo:

dias= [‘lunes’,’martes’,’miercoles’,’jueves’,viernes’,’sabado’,’domingo’]

Numero de orden	Valor
0	lunes
1	martes
2	miercoles
3	jueves
4	viernes
5	sabado
6	domingo

Listas

- Acceso a un elemento:
  - dias[3] el número de orden comienza en 0

```
>>> dia[3]
jueves
```

  - Si el índice es negativo se empieza a contar desde el último elemento comenzando por -1

```
>>> dia[-1] # Negativo, desde el final
domingo
```

0	lunes
1	martes
2	mercredi
3	jueves
4	sabado
5	domingo

Listas

- Modificación de un elemento de la lista
  - dias[2]='mercredi’
- Troceado (slicing): Se usa para acceder a rangos de elementos en vez de a elementos individuales.
  - Devuelve otra lista con los elementos que van desde el primer índice al último menos 1 (excluye el último).
  - lista[inicio:fin:incremento]

```
>>> numeros=[1,2,3,4,5,6,7,8,9,10]
>>> numeros[4:7]
[5, 6, 7]
```

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

```
>>> numeros[:3] #Excluye el ultimo
[1, 2, 3]

>>> numeros[::2] # Extrae cada dos elementos
[1, 3, 5, 7, 9]
```

Listas

- Las listas pueden contener elementos de distinto tipo
  - Lista = [1,'azul',23,'casa',True]
- Las listas se pueden anidar

```
lista = [[1,2,3],[3,1],2]
```

- Acceder a una lista anidada

```
>>> lista[0][1]
2
```

- Indices fuera de rango dan error

  Lista[5] ←ERROR

- Podemos obtener la longitud de la línea con la función len()

```
>>> len(lista)
        3
>>> len(lista[1])
        2
```

Listas

- El troceado también se puede hacer con cadenas

Ejercicio:

Dada la lista lista = ['Mr.', 'John', 'Smith', 34, 1.73]

- Extraer el primer elemento
- Extraer el último elemento
- Extraer el cuarto elemento, de dos formas
- Extraer los elementos centrales (excluir primero y último)
- Reemplazar el primer elemento por una lista con los dos últimos elementos

Listas

Ejercicio . Solución

Dada la lista: lista = ['Mr.', 'John', 'Smith', 34, 1.73]

- Extraer el primer elemento

  lista[0]
- Extraer el último elemento

  lista[-1]
- Extraer el cuarto elemento, de dos formas

  lista[3], lista[-2]
- Extraer los elementos centrales (excluir primero y último)

  lista[1:-1]
- Reemplazar el primer elemento por una lista con los dos últimos elementos

  lista[0]=lista[3:] ó lista[0] = lista[len(lista)-2:]

Tuplas

- Son un tipo colección que no puede modificarse. Inmutables
- Una serie de elementos separados por comas forman una tupla. Habitualmente se encierra entre paréntesis

```
>>> tupla = (1,2,3)
```

- Veremos su utilidad cuando estudiemos funciones
- También sirve para asignar valores a varias variables:

```
a,b= (1,2)
```

- La función tuple convierte una lista o una cadena en una tupla

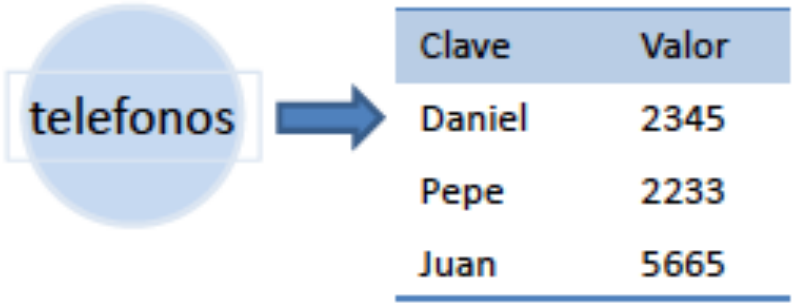
```
>>> tuple([1,2,3,4])
(1, 2, 3, 4)
>>> tuple("abc")
('a', 'b', 'c')
```

- Las tuplas soportan muchas de las acciones de las cadenas

Diccionarios

- Tipo colección
  - Son conjuntos no ordenados de pares clave-valor
  - Se definen poniendo entre {} la lista separada por , de elementos clave:valor separados por :
  - Los valores pueden ser de cualquier tipo, incluido otro diccionario

```
>>>telefonos = {'Daniel' : 2345, 'Pepe' : 2233, 'Juan' : 5665}
```



Diccionarios

- Acceso por clave
  - Dato= telefonos[‘Daniel’]

Accede al valor asociado a la clave Daniel

- Modificación de un elemento
  - telefonos[‘Daniel’]=9898
- Lista de las claves
  - telefonos.keys()
- Lista de valores
  - Telefonos.values()

Diccionarios

- Ejercicio

- Escribir un diccionario que tenga como claves los numeros en español del 1 al 4 y como valores sus nombres en inglés
- Acceder al elemento ‘dos’ y almacenarlo en otra variable.
- Imprimir todas las claves y valores

Diccionarios

- Solución

```
Dic={"uno":"one","dos":"two","tres":"three","cuatro":"four"}
v=Dic["dos"]
print"Estas son las claves:",Dic.keys(),"\n"
print"Estos son los valores:",Dic.values(),"\n"
```

Sentencias de control

- Bloque
  - Un bloque es un grupo de sentencias que se ejecutan si una condición se cumple. Puede ser parte de una sentencia condicional o de un bucle
  - Se crea un bloque al indentar las sentencias que se quiere formen parte del bloque
  - Los bloques comienzan con **:** y finalizan cuando la siguiente sentencia tiene menor indentación.

```
Esto es una sentencia
Esto es otra:
    esta es la primera línea del bloque
    esta la segunda

Esta ya no pertenece al bloque
```

Sentencias condicionales

- Valores booleanos:
  - Falso: False, None, 0, “”, (), [], {}
  - Verdadero: Todo lo demás
  - Operadores booleanos: and y or
- Operadores de comparación

Expresión	Descripción
x == y	Igual
x < y	Menor que
x > y	Mayor que
x <= y	Menor o igual
x >= y	Mayor o igual
x != y	Distinto que
x is y	x e y son el mismo objeto
x is not y	x e y son diferentes objetos
x in y	x es un miembro de y
x not in y	x no es un miembro de y

Sentencias condicionales

- **Sentencia if**

  if condicion:

  bloque

- **Sentencia if-else**

  if condicion:

  bloque

  else

  bloque

- **Sentencia if-elif-else**

  if condicion:

  bloque

  elif condicion:

  bloque

  …

  else:

  bloque

```
num=input("Teclea un numero por pantalla\n")
if num > 0:
    print " El numero es mayor que 0"
elif num < 0:
    print"El numero es menos que 0"
else:
    print"El numero es igual a 0"
```

Bucles

Bucles while:

while condición:

bloque

```
i=0
while i<4:
    i+=1
    print i
```

La variable i se incrementa en 1 en cada iteración

Bucles for:

for i **in** colección: #obligatorio
  bloque #indentado

Los bucles for se pueden anidar:

for i **in** *colección:*
  for j **in** *colección:*
    Instrucciones

Se puede romper la ejecución de un bucle con **break**
Se puede interrumpir un ciclo y pasar al siguiente con **continue**

Bucles

Ejemplos

```
asistentes=["rosa","lucia","clara"]
for i in asistentes:
    print i
```

```
for x in range(1, 4):
    for y in range(1, 4):
        r=x*y
        print x,"*",y,"=",r
```

Range(i,j) devuelve una lista de valores entre i y j-1

Ejercicio:
Crea una lista de 10 numeros de valores entre 0 y 50. Recorre la lista con un bucle e imprime por pantalla aquellos mayores que 5

Bucles

Solución

```
l=[34,5,45,5,3,8,43,2,1,10]
for i in l:
    if i>5:
        print i
```

Bucles

EJERCICIO PARA SUBIR AL CAMPUS

Pedir al usuario una cadena de ADN y contar cuántas veces aparece cada una de las bases. Imprimir finalmente el numero de veces que aparece cada base.

NOTA: en vez de utilizar la función input para pedir al usuario la cadena por pantalla utilizar la función raw_input()

Modulos

- La 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

- Ejercicio solución

- Buscar como hacer senos, cosenos y exponenciales
- **math.sin(1), math.cos(1), math.exp(1)**

- Logaritmos neperianos y decimales
- **math.log(1), math.log10(1)**

- Redondeos al entero superior, al inferior y al más cercano
- **math.ceil(1.2), math.floor(1.6), round(1.6)**

- Valores absolutos
- **abs(-2), math.fabs(-2)**

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

```
>>> 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 numeros enteros**
    - **np.arange()**

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
>>> np.arange(6,12,2)
 array([ 6,  8, 10])
```

(inicio,fin, incremento)

Se excluye el último

  - **Vector con rango de valores reales**
    - **np.linspace()**

Vector de 10 valores entre 0 y 1

```
>>> np.linspace(0,1,10)
array([ 0. , 0.11111111,  0.22222222,  0.33333333,  0.44444444,
0.55555556,  0.66666667,  0.77777778,  0.88888889,  1.])
```

Modulos: numpy

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

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

Modulos numpy

- Ejercicio
  - Crear los siguientes arrays :
  - Vector con números del 0 al 30, 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

- Solución
- Vector con números descendentes del 0 al 30, de 3 en 3
- **np.arange(0,30,3)**
- Matriz identidad 5x5: **np.identity(5)**
- Matriz con números del 1 al 5 en la diagonal
- **a = np.diag(np.arange(1,6))**
- Lo mismo pero del 2 al 10, de 2 en 2: **a*2**
- ¿Qué submódulo...? **numpy.random**
- Hacer una matriz de 2x3 de números aleatorios
- **np.random.rand(2,3)**