Python (Seconda parte)

Laboratorio di Metodi Computazionale e Statistici (2022/2023)

Fabrizio Parodi and Roberta Cardinale

Dipartimento di Fisica

Funzioni

Funzione simile alla void in C++

```
def test():
        print('in test function')
test()
```

```
in test function
```

Funzione con valore di ritorno

```
def area_disco(r):
        return 3.14*r*r
area_disco(1.5)
```

```
7.064999999
```

Funzioni con parametri default

```
def square(x=2):
        return x*x
square()
square(3)
```

```
4
9
```

Funzioni parametri indirizzati per nome

```
def square(x=2,y=3,z=4):
        return x+y+z
square()
square(1)
square(1,2)
square(1,z=2)
square(x=1,y=3,4) # invalido !
```

- I parametri "posizionali" devono precedere quelli "named"
- Una volta specificati parametri con nome non ne possono più essere passati come posizionali

Funzioni con parametri multipli

```
def multiply(*args):
    z = 1
    for num in args:
        z *= num
    print(z)
multiply(1,2,3)
multiply(1,2)
```

```
6
2
```

Funzioni con più valori di ritorno

```
def yfunc(t, v0):
    g = 9.81
    y = v0*t - 0.5*g*t**2
    dydt = v0 - g*t
    return y, dydt
position, velocity = yfunc(0.6, 3)
print(position, velocity)
```

Duck typing

Gli argomenti sono passati per assegnazione: in Python le variabili sono riferimenti ad oggetti; nel passaggio degli argomenti, alla variabile nella funzione viene assegnato lo stesso oggetto della variabile corrispondente nella chiamata; cioé viene fatta una copia dei riferimenti agli oggetti (quelli mutabili possono essere "mutati", gli altri no)

I tipi delle variabili vengono definiti solo all'assegnazione dei riferimenti, per cui una funzione, a priori, non sa quali sono i tipi degli argomenti ed eventuali inconsistenze producono errori solo quando la funzione viene eseguita.

In questo modo Python implementa naturalmente il polimorfismo, cioe' una stessa funzione puo' essere usata per dati di tipo diverso.

"If it walks like a duck, and quacks like a duck, its a duck"

Duck typing

```
def sum_of(*items):
    result = items[0]
    for item in items[1:]:
        result = result + item
    print(result)
```

```
sum_of(2, 3, 5)         # -> 10
sum_of("2", "3", "5") # -> "235"
sum_of([2], [3, 5])    # -> [2, 3, 5]
sum_of(2 , b)           # TypeError
```

Gestione della memoria

- Al contrario del C++ Python gestisce autonomamente la memoria
- I seguenti oggetti sono unici (vengono creati autonomamente da Python)
  - interi tra -5 e 256
  - alcune stringhe
  - contenutori immutabili vuoti (tuples)
- Ogni altro oggetto viene creato quanto si assegna un "nome" (o una referenza) ad esso o quando viene passato ad una funzione.
- Quando una variabile non ha più "nomi" (referenze) che si riferiscono ad essa (reference count) la corrispondente porzione di memoria viene cancellata (garbage collection)

Variabili globali e locali

Regola generale: quando ci sono più variabili con lo stesso nome python prima cerca variabili locali, poi quelle globali poi le funzioni/variabili di sistema.

```
numbers = [2.5, 3, 4, -5]
print(sum(numbers))     # built-in Python func.
sum = 500               #  sum e' int (globale)
print(sum)
def myfunc(n):
    sum = n + 1
    print(sum)          # sum e' locale
    return sum
sum = myfunc(2) + 1     # update var. globale sum
print(sum)
```

Variabili globali e locali

Si può forzare l'uso di variabili globali con la keyword `global`

```
a = 2; b = 1      # globali

def f1(x):
    return a*x + b

def f2(x):
    global a
    a = 3
    return a*x + b

f1(2); print(a)       #stampa   2
f2(2); print(a)       #stampa   3
```

Commenti in python

I commenti in Python si possono mettere in due modi

- linea di commento

```
# questa e' una linea di commento
```

- regione commentate

```
"""
tutta
questa
regione
e' commentata
"""
```

Classi

- La forma più semplice di definizione di una classe è del tipo:

```
class NomeClasse:
    <istruzione-1>
    .
    .
    .
    <istruzione-N>
```

- L'operazione di instanziazione (la "creazione" di un oggetto classe) crea un oggetto vuoto. In molti casi si preferisce che vengano creati oggetti con uno stato iniziale noto. Perciò una classe può definire un metodo speciale chiamato `__init__()`, come in questo caso:

```
    def __init__(self):
        self.data = []
```

Classi

- tutti i metodi devono avere come primo argomento l'istanza stessa (che alla chiamata non viene passata)

```
    def func(self,arg):
        self.variabile = arg
```

Esempio: classe vector

```
class vector:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def __add__(self,b):
        return vector(self.x+b.x,self.y+b.y)
    def __str__(self):
        return "("+str(self.x)+","+str(self.y)+")"

a = vector(2,2)
b = vector(3,3)
c = a+b
print(c)
```

Ereditarietà

Anche in Python si può ereditare una classe derivata da una classe base.
La sintassi è:

```
class ClasseDerivata(ClasseBase):
```

Esempio: classe complex

```
class vector:
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def __add__(self,b):
        return vector(self.x+b.x,self.y+b.y)
    def __str__(self):
        return str(self.x)+","+str(self.y)

class complex(vector):
    def cong(self):
        return complex(self.x,-self.y)

c = complex(1,2)
d = c.cong()
print(d)
```

I/O da terminale

Input

```
a = input('numero ?')
print(a)
```

Output

```
a = 10
print(a)
```

I/O da file

Input

```
f = open('workfile', 'r')
for line in f:
     print(line)
```

Output

```
f = open('workfile', 'w')
f.write('Test\n')
a=10
f.write(repr(a))
```

`repr` fornisce una rappresentazione di a sottoforma di stringa.

Moduli: come importarli

Semplice importazione di un modulo (caricamento di una certa libreria)

```
import sys
```

Importazione con cambio del nome

```
import sys  as system
```

Importazione di una sola parte del modulo

```
from functools import lru_cache
```

Importazione di tutto (senza bisogno di specificare il nome del modulo)

```
from os import *
```

Attenzione ! Se definite una funzione con lo stesso nome di una presente nel modulo "sovrascrivete" quella del modulo (perché viene usato senza utilizzarne il nome).

Moduli: sys

test.py:

```
import sys
print(sys.argv)
```

```
> ./test.py test arguments
[file.py, test, arguments]
```

Moduli: os

test.py:

```
import os
os.listdir('.')
```

```
[ 'inherit4.jpg', 'introduzione.aux', 'introduzione.log', 'introduzi
```

Moduli: numpy

numpy è un modulo con molte oggetti "numerici" utili.
Ad esempio array:

```
import numpy as np;
a = np.array([1, 2, 3])  # Create a rank 1 array
print(type(a))  # Prints "<type 'numpy.ndarray'>"
print(a)        # Prints "(3,)"
a[0] = 5        # Change an element of the array
print(a)        # Prints "[5, 2, 3]"
```

Come list ma con calcolo vettoriale. Altro esempio

```
import numpy as np
x1 = np.linspace(0,100,100)
x2 = np.zeros(100, float)
x3 = np.ones(100,float)
x4 = np.ones(100,float)*5
```

Moduli: numpy

Creazione array vuoto, aggiunta di un elemento alla volta

```
import numpy as np;
a = np.array([])
for i in range(0,10):
    a = np.append(a,i)
print(a)
```

Moduli: ROOT

Tra gli altri si può usare anche ROOT da Python

```
#!/bin/python
import math
from ROOT import gApplication, TCanvas, TF1, TMat
fun1 = TF1('fun1','sin(x)/x', -5*math.pi,
                               5*math.pi )
fun1.Draw()
gApplication.Run()
```

Moduli: Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-5*np.pi,5*np.pi,100)
y = np.sin(x)/x
plt.plot(x,y)
plt.show()
```

Info, esempi e guide su `https://matplotlib.org/users/index.html`

Moduli: Matplotlib

La sintassi generale di plot è la seguente:

```
plot([x], y, [fmt])
```

dove `[x]` e `[fmt]` indicano parametri che possono essere omessi.
I possibili valori (concatenabili !) per la stringa `[fmt]` sono i seguenti:

Line Styles

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

Colors

The supported color abbreviations are the single letter codes

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

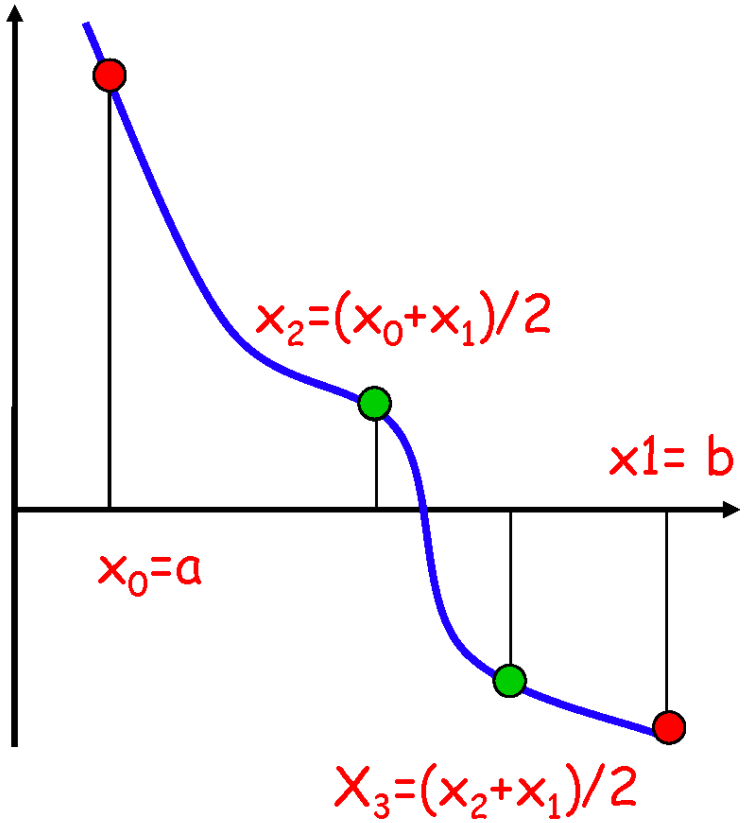
Implementazione in Python dell'algoritmo di bisezione

Il metodo di bisezione per la ricerca degli zeri consiste nel fissare un intervallo (ai cui estremi la funzione ha segno opposto), calcolare il valore della funzione nel punto medio e determinare in quale dei due sotto-intervalli si trova lo zero.
Ovvero si controlla quale coppia tra

`f(x0,x1)` e `f(x1,x2)`

sia di segno discorde.

L'intervallo che contiene lo zero diventa il nuovo intervallo di ricerca e così via. L'iterazione continua fino a che l'ampiezza dell'intervallo non scende sotto un valore prefissato.



Notebook

- Python (così come altri linguaggi) può essere runnato in particolari interfacce web Jupyter (Julia/Python/R) Notebook
  - Il notebook può basarsi su risorse locali (useremo questo)
  - O utilizzare software su server remoti
- Utilizzeremo il notebook di ROOT (che supporta C++, Python e ROOT)
- Il notebook è un tool di discussione/sviluppo/didattica. Permette di presentare testo e codice insieme.
- Estremamente utile per "abbozzare" un progetto, meno per il suo sviluppo.

Notebook

- Per installare

```
python3 -m pip install notebook
```

- In realtà voi lo avete già incluso nella vostra installazione (installWSL)
- Dovete però avere un browser installato e c'è un piccolo fix per Win10/11

```
sudo apt install firefox
python3 -m pip install aws-sam-cli
```

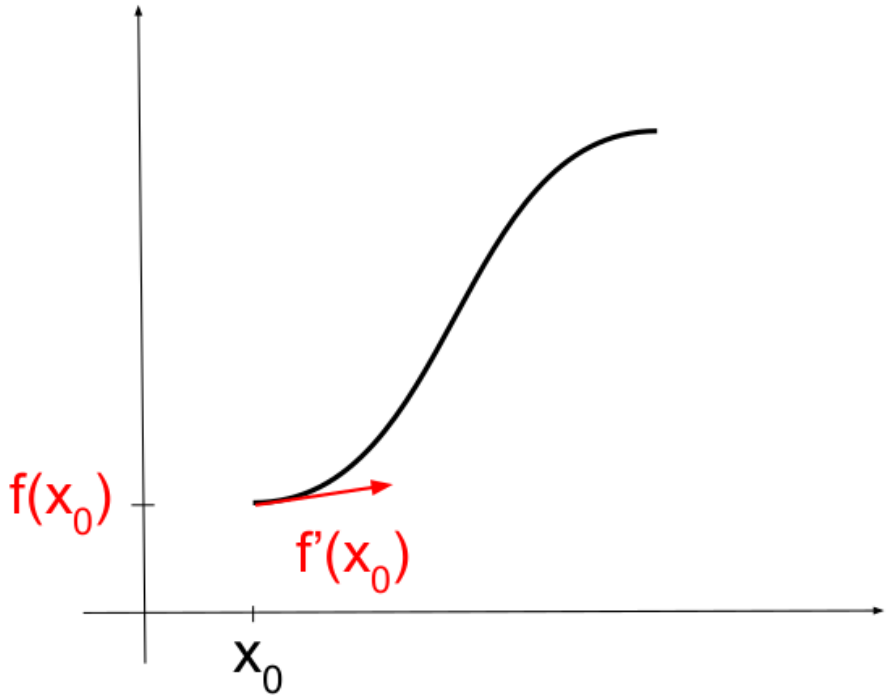
- Per far partire il notebook (su browser)

```
root --notebook
```

Applicazioni: equ. differenziali con condizioni al contorno

Finora abbiamo visto equazioni differenziali con condizioni iniziali

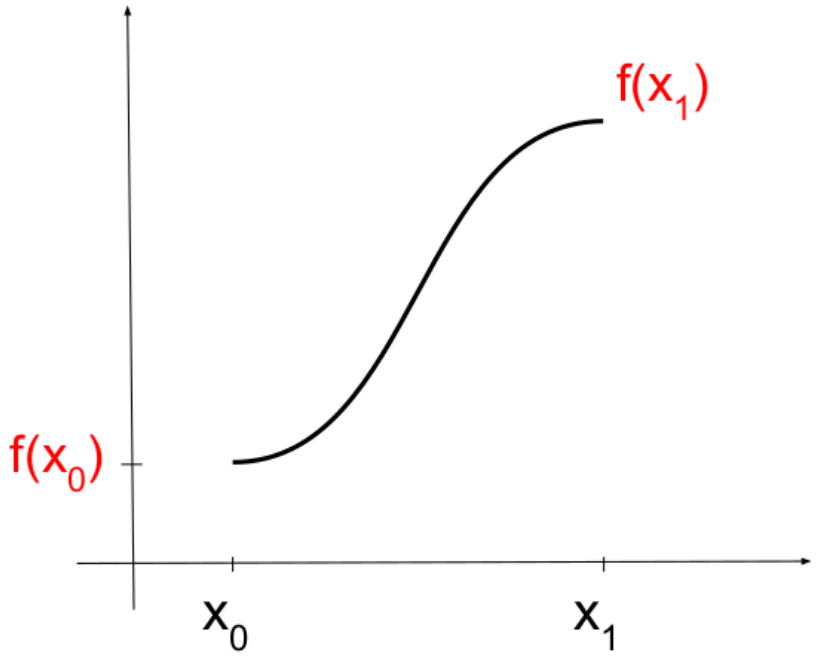
$$\frac{d^2 f}{dx^2} = g(f', f, x)$$
$$f(x_0) = \alpha$$
$$f'(x_0) = \beta$$



Applicazioni: equ. differenziali con condizioni al contorno

In alcuni casi, tuttavia, sono fornite condizioni al contorno in un intervallo $[x_0, x_1]$ (sono noti i valore di f agli estremi ma non la sua derivata nel punto x_0)

$$\frac{d^2 f}{dx^2} + k(x) f(x) = 0$$
$$f(x_0) = \alpha$$
$$f(x_1) = \beta$$



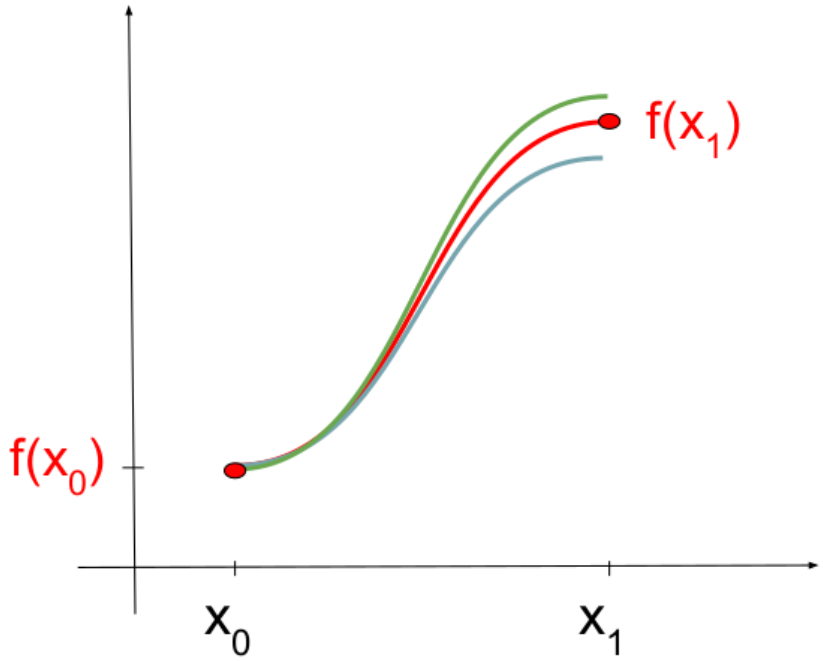
Possiamo applicare i metodi visti applicando una procedura lievemente diversa, lo "Shooting method":

- se conosco $f(x_0)$ posso far variare la derivata in x_0 (e quindi la soluzione) finché la soluzione evolve fino a essere uguale a $f(x_1)$ in x_1

Applicazioni: equ. differenziali con condizioni al contorno

In alcuni casi, tuttavia, sono fornite condizioni al contorno in un intervallo $[x_0, x_1]$ (sono noti i valore di f agli estremi ma non la sua derivata nel punto x_0)

$$\frac{d^2 f}{dx^2} + k(x) f(x) = 0$$
$$f(x_0) = \alpha$$
$$f(x_1) = \beta$$



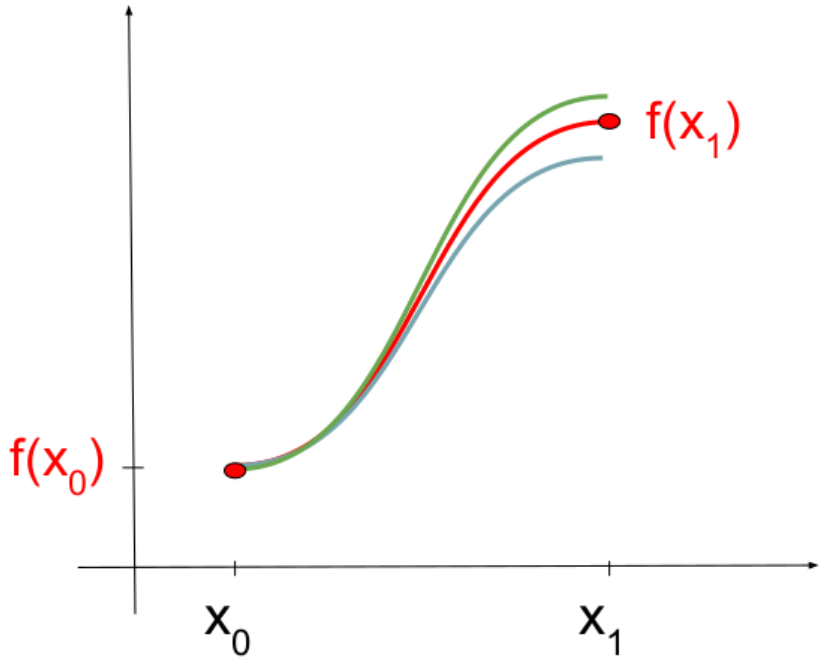
Possiamo applicare i metodi visti applicando una procedura lievemente diversa, lo "Shooting method":

- se conosco $f(x_0)$ posso far variare la derivata in x_0 (e quindi la soluzione) finché la soluzione evolve fino a essere uguale a $f(x_1)$ in x_1

Applicazioni: equ. differenziali con condizioni al contorno

In alcuni casi, tuttavia, sono fornite condizioni al contorno in un intervallo $[x_0, x_1]$ (sono noti i valore di f agli estremi ma non la sua derivata nel punto x_0)

$$\frac{d^2 f}{dx^2} + k(x)f(x) = 0$$
$$f(x_0) = \alpha$$
$$f(x_1) = \beta$$



Possiamo applicare i metodi visti applicando una procedura lievemente diversa, lo "Shooting method":

- se conosco $f(x_0)$ posso far variare la derivata in x_0 (e quindi la soluzione) finché la soluzione evolve fino a essere uguale a $f(x_1)$ in x_1
- il problema può essere "automatizzato" tramite una ricerca di zeri della funzione

$$f_{evol}(x_1 | f'(x_0), f(x_0)) - f(x_1) = 0$$

dove $f'(x_0)$ è la "variabile" della ricerca di zeri.