Elaborazione di Segnali Multimediali

Introduzione a Python

D.Cozzolino

Questa è una breve introduzione al Python che ha l'obiettivo di fornirvi gli elementi essenziali per il suo utilizzo. Chi è interessato ad una guida completa può riferirsi alle documentazione ufficiale al sito `https://docs.python.org/3/` o ai tutorial offerti dalla w3schools `https://www.w3schools.com/python` Leggete questo documento e provate tutte le istruzioni che troverete di seguito.

1 Concetti base

Lanciando Spyder, avete a disposizione la console Interattiva di Python (Interactive Python, IPython), da cui è possibile digitare i comandi. Per esempio, si può usare la console come se fosse una semplice calcolatrice, digitando

```
In [1]: 4+5*2
```

e premendo INVIO otterremo la risposta.

```
Out [1]: 14
```

Provate adesso ad usare il comando `print`, eseguendo la seguente istruzione

```
In [2]: print("Hello, World!")
```

Il comando `print` scrive in console il testo fornito tra le parentesi, in questo caso "Hello, World!".
In Python, è possibile avere un guida per ogni comando e tipo usando l'istruzione `help`. Provate ad ottenere l'help in linea per il comando `print` eseguendo

```
In [3]: help(print)
```

Usate sempre l'help in linea quando avete dei dubbi su come deve essere usato un determinato comando.

2 Variabili

I risultati dell'operazione possono essere memorizzati in Variabili tramite l'operatore *uguale* (=). Provate ad eseguire le seguenti istruzioni:

```
a = 10*2
c = "ALOHA!"
```

Nell'esempio precedente, `a` e `c` sono le variabili che adesso memorizzano rispettivamente il numero 20 e la scritta ALOHA!. E' importante sottolineare che in Python non è necessario specificare il tipo della variabile che risulta automaticamente definita in seguito all'assegnazione dei valori che deve assumere. Digitando il nome della variabile è possibile conoscerne il valore.

```
In  [ ]: a
Out [ ]: 20
```

I nomi delle variabili possono contenere solo lettere, numeri e il carattere underscore (_) e non possono iniziare con un numero.
E' possibile cancellare una variabile tramite il comando `del`. Prova il seguente codice:

```
f = "roses are red"
print(f)
del(f)
print(f)
```

L'elenco di tutte le variabili usate nella console sono visibili nel Variable Explorer di Spyder. In alternativa è anche possibile visualizzare l'insieme di tutte le variabili, e il loro tipo, della console attraverso il seguente *magic command*:

```
In  [ ]: %whos
```

I *magic command* iniziano con il carattere % e sono esclusivi della console Interattiva di Python (Interactive Python, IPython) e quindi non possono essere utilizzati negli script e nei moduli Python. Un altro *magic command* utile è `%reset` che resetta la console cancellando tutte le variabili.

2.1 Tipi base

Python ha diversi tipi tra cui: i numeri interi (`int`), i numeri a virgola mobile (`float`), le stringe (`str`) e i booleani (`bool`).

```
a = 10          # un intero
b = 3.14        # un numero floating point
c = "string"    # una stringa
d = True        # uno booleano
```

La funzione `int( )` converte l'input in un intero. Analogamente, le funzioni `str( )` e `float( )` convertono l'input rispettivamente in una stringa e in un numero a virgola mobile. Per sapere il tipo di una variabile si può usare la funzione `type( )`. Mentre, possiamo usare la funzione `isinstance( )` per verificare se la variabile è di un dato tipo. Provate ad usare le funzioni appena descritte.

```
a = 5
b = str(a)
isinstance(a, int)
isinstance(b, int)
type(b)
```

Python prevede diverse operazioni tra i tipi numerici (tabella 1). Inoltre, è possibile usare le parentesi tonde per definire l'ordine delle operazioni. Provate le seguenti istruzioni:

Operazione	Descizione
+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
//	Divisione arrotondata per difetto
%	Modulo
**	Elevazione a potenza
abs( )	Valore Assoluto
==	Restituisce **True**, se sono uguali
!=	Restituisce **True**, se non sono uguali
<	Restituisce **True**, se il primo termine è minore del secondo
<=	Restituisce **True**, se il primo termine è minore o uguale del secondo
>	Restituisce **True**, se il primo termine è maggiore del secondo
>=	Restituisce **True**, se il primo termine è maggiore o uguale del secondo
and	Restituisce **True**, se entrambi i termini sono **True**
or	Restituisce **True**, se almeno uno è **True**
not	Restituisce **True**, se il termine è **False**

Tabella 1: Operazioni per tipi numerici

```
a = 5
b = 3
a+b * 5
(a+b)*5
```

Python prevede due tipi di divisioni con *singolo slash* (/) o *doppio slash* (//). Nel caso si usi un *singolo slash* il risultato della divisione tra due `int` è un `float`. Mentre, bisogna usare il *doppio slash* per ottenere un intero.

```
In  [ ]: 7 / 4
Out [ ]: 1.75
In  [ ]: 7 // 4
Out [ ]: 1
```

N.B. Un'utile funzionalità di Spyder è l'uso del tasto "freccia in alto" (↑), che permette di ridurre notevolmente i tempi di immissione, infatti premendo questo tasto si ottengono le istruzioni precedentemente digitate, che si possono eventualmente modificare; lo stesso comando può agire in maniera più selettiva, digitando una lettera o un gruppo di lettere e poi ↑, vengono riproposti solo i comandi che iniziano con la lettera o il gruppo di lettere digitati.

2.2 Le Liste

Python nativamente include tipi strutturati per raccogliere o raggruppare dati. Noi ci concentreremo solo sulle liste di Python che permettono di gestire una sequenza di elementi dove ogni elemento può essere di un tipo differente. Possiamo definire una lista usando le parentesi quadre o usando la funzione `list`.

Operazione	Descrizione
+	Concatena due liste
len( )	Per ottenere la lunghezza della lista
*	per replicare una lista
in	determina se un elemento è nella lista
not in	determina se un elemento non è nella lista

Tabella 2: Operazioni per le liste

```
b = [4, 6, 'cat', 2.71]
c = [7.8, ['string', 3.24]]
lista_vuota = list()
```

Nota che un elemento di una lista può essere un'altra lista. Le parentesi quadre vengono anche utilizzate per indicizzare l'elenco, è importante ricordare che l'indicizzazione inizia da 0 in Python. Inoltre, il valore dell'indice può anche essere negativo. Usando il valore dell'indice -1 otteniamo l'ultimo elemento della lista. Usando il valore dell'indice -2 otteniamo il penultimo elemento della lista e così via.

```
b = [4, 6, 'cat', 2.71]  # lista di 4 elementi
b[1]     # secondo elemento della lista
b[-1]    # ultimo elemento della lista
b[2]     # terzo elemento della lista
```

Possiamo estrarre una parte della lista usando la sintassi [start:stop:step]. La lista estratta conterrà gli elementi partendo da quello con indice start con un passo pari a step fino all'elemento con indice stop -1. Nota che l'elemento con indice stop non è incluso. Lo step può essere omesso usando la sintassi [start:stop], in questo caso lo step è uguale a 1. Possiamo anche omettere uno (o entrambi) gli start e stop, il default è rispettivamente l'inizio e la fine della lista.

```
x = [ 1, '2', 3.0, 4, '5'] # a list of 5 elements
x[2:4]   # sottolista di due elementi a partire dal terzo elemento
x[3:]    # sottolista senza i primi tre elementi
x[:-3]   # sottolista senza gli ultimi tre elementi
x[::2]   # estrae gli elementi con valori di indice pari
```

La lista può essere modificata, infatti possiamo cambiare il valore di un elemento della lista semplicemente usando le parantesi quadre per accedere a quell'elemento.

```
b = [4, 6, 'cat', 2.71]
b[1] = 11.56         # possiamo modificare gli elementi della lista
```

Alcune delle operazioni applicabili alle liste sono riportate in tabella 2, notate che l'operatore *più* (+) serve per concatenare le liste. Di seguito riportiamo un esempio:

```
In  [ ]: a = [0, 4, 4]
In  [ ]: b = [5, 9, 8]
In  [ ]: a + b
Out [ ]: [0, 4, 4, 5, 9, 8]
```

Python prevede anche le tuple che sono simili alle liste, ma non posso essere modificate dopo essere state create. Possiamo definire una tupla usando le parentesi tonde.

```
b = (4, 6, 'cat', 2.71)
b[1] = 11.56  # this is not allowed
```

3 Script e costrutti di programmazione

Per eseguire un blocco di istruzioni è necessario creare uno *script* Python, cioè un file con estensione .py nel quale siano inserite tutte le istruzioni che si vogliano eseguire. Spyder include un editor per creare e modificare i file con estensione .py. Per lanciare lo script basta premere sull'icona "Run File" o eseguire nella console il *magic command* %run seguito dal nome del file. Proviamo a creare un primo *script* usando l'editor di Spyder

Listing 1: my_script.py

```
print("My first Script")
```

e eseguiamolo scrivendo nella console:

```
In [ ]: %run my_script.py
```

Durante le esercitazioni provate sempre a creare degli script piuttosto che a digitare in console, in questo modo risparmierete tempo quando volete ripetere e/o correggere dei comandi.
Python prevede i classici costrutti di programmazione come il costrutto if-else.

```
a = 4
if a>10:
    print("viene eseguito solo se 'a' è maggiore di 10")
print("viene eseguito sempre perchè fuori dal blocco IF")
```

Notate che il blocco di istruzione in Python si definisce tramite l'indentazione, se l'indentazione non è corretta il programma è sbagliato.

```
a = 4
if a>10:
    print("viene eseguito solo se 'a' è maggiore di 10")
    print("incluso del blocco IF")
```

Un altro costrutto classico è il for con cui possiamo scorrere una lista di interi definita dalla funzione range(start,stop,step) che ha una sintassi simile indicizzazione delle liste:

```
# scrive i numeri da 0 a 9
for x in range(10):
    print(x)
```

oppure possiamo scorrere una lista predefinita:

```
l = ["dog", "cat", "mouse"]
# scrive gli elementi della lista
for x in l:
    print(x)
```

4 Funzioni, Moduli e Package

In Python si possono creare delle funzioni che realizzano le operazioni di cui abbiamo bisogno e che non sono già presenti. Facciamo subito un esempio scrivendo in uno script la funzione (banale) che restituisce la metà di un valore.

```
def half(x):
    """
    Restituisce la metà del valore dato
    """
    return x / 2

print(half(10))
```

Inoltre digitando `help(half)` verrà visualizzato tutto ciò che avete scritto nel commento delineato dai caratteri """, questo è utile per descrivere il comportamento della funzione. Cercate quindi di documentare sempre e in modo appropriato le funzioni che scrivete in modo che possano essere utilizzate facilmente. Le funzioni possono essere usate anche esternamente allo *script*. Infatti, ogni file .py può essere considerato come un *modulo* Python, cioè una collezione di funzioni, variabili ed altro. Crea il file *my_module.py* con il seguente codice:

Listing 2: my_module.py

```
"""
Esempio di modulo python. Contiene una variabiel chiamata
my_variable e una funzione chiamata my_function.
"""

my_variable = 0

def my_function():
    """
    Funzione di esempio
    """
    return my_variable
```

Un *modulo* Python può diventare accessibile dalla console o da un altro modulo usando la parola chiave `import`.

```
import my_module
```

Possiamo usare le variabili e le funzioni presenti nel modulo attraverso l'operatore punto (.)

```
a = my_module.my_variable
b = my_module.my_function()
```

Possiamo anche definire un alias al modulo usando la seguente sintassi:

```
import my_module as alias
```

I *moduli* Python possono essere raggruppati in *package* Python. Un *package* Python è semplicemente una directory che contiene i file .py dei vari *moduli* inclusi e un file per l'inizializzazione chiamato `__init__.py`. Un package standard di Python è **os** che è usato principalmente per gestire i file. Di seguito sono riportati vari esempi di come importarlo.

```
import os                               # importa il package
a = os.path.isfile('my_module.py')

from os import path                     # importa il modulo
a = path.isfile('my_module.py')

from os.path import isfile              # importa solo la funzione
a = isfile('my_module.py')
```

5 Numpy per vettori e matrici

Vettori a matrici non sono gestiti nativamente in Python, per questo è necessario l'utilizzo della libreria Numpy. Il primo passo è importare il modulo *numpy* usando l'alias *np* per comodità.

```
import numpy as np
```

La libreria Numpy prevede il tipo N-dimensional-array che utilizzeremo per vettori e matrici. Rispetto ad un lista gli elementi di un Numpy array devono essere omogenei. In altre parole tutti gli elementi di un array devono essere dello stesso tipo. Un array può essere creato a partire da una lista usando la funzione `np.array`:

```
V = np.array([1,2,3,4,5,6])
```

In questo modo abbiamo definito un vettore di sei elementi con valori 1,2,3,4,5 e 6. Analogamente per le matrici possiamo applicare la funzione `np.array` a una lista di liste:

```
A = np.array([[1,2,3], [4,5,6]])
```

Otteniamo così la variabile `A` che è una matrice 2×3: la prima riga ha i valori $1, 2, 3$ e la seconda i valori $4, 5, 6$.
Per accedere ad un elemento della matrice possiamo fornire gli indici tramite parentesi quadre.

```
a = A[0,2]
```

Ricordatevi sempre che in Python gli incidi partono da 0.
E' anche possibile estrarre sotto-matrici da una matrice utilizzando l'operatore ":". Supponiamo per esempio di aver definito una matrice D di dimensioni 9×8; per estrarre delle sottomatrici da D si possono usare i seguenti comandi:

`D[1:5,0:3]`
estrae una matrice 4×3 da D le cui righe vanno dalla seconda alla quinta, mentre le colonne dalla prima alla terza.

`D[1,:]`
estrae la seconda riga dalla matrice; per estrarre la seconda colonna basta digitare `D[:,1]`

Per convertire il tipo degli elementi di un array (chiamato `dtype`) possiamo usare il metodo `astype()`.

```
A.astype(np.int16)
```

Possibili `dtype` previsti da Numpy sono: `np.uint8`, `np.int64`, `np.float32`, `np.float64`, `np.complex64` e `np.bool`. La funzione `np.array` determina automaticamente il `dtype` analizzando gli elementi della lista. In alternativa il `dtype` può anche essere esplicitato nel seguente modo:

```
V = np.array([1,2,3], dtype=np.float32)
```

Di seguito si elencano alcune funzioni molto utili per la definizione di matrici e vettori:

`A = np.zeros((N,M))`
definisce una matrice di `N` righe ed `M` colonne con elementi tutti nulli. Se non è esplicitato il `dtype` è `np.float64`.

`A = np.ones((N,M))`
definisce una matrice di `N` righe ed `M` colonne con elementi tutti unitari. Se non è esplicitato il `dtype` è `np.float64`.

`B = np.copy(A)`
definisce l'array `B` replicando l'array A. Fate attenzione ad evitare assegnazioni dirette del tipo `B = A`, perché in questo caso `B` e `A` risulterebbero lo stesso array e qualsiasi manipolazione fatta su variabile `B` risulta anche su variabile `A`.

`A = np.eye(N)`
definisce una matrice identità quadrata di `N` righe ed `N` colonne.

`c = np.arange(N)`
definisce un vettore coi numeri fra 0 a N-1 con passo 1.

`c = np.arange(start,stop,step)`
definisce un vettore che parte del valore `start` con un passo pari a `step` fino al valore `stop`-1. Nota che il valore `stop` non è incluso.

`B = A.T`
definisce B come trasposta di A.

`C = np.hstack((A,B))`
definisce la matrice `C` affiancando le due matrici `A` e `B`, che devono avere lo stesso numero di righe.

`C = np.vstack((A,B))`
definisce la matrice `C` sovrapponendo le due matrici `A` e `B`, che devono avere lo stesso numero di colonne.

5.1 Operazioni sugli array

Sugli array si possono eseguire operazioni elemento per elemento, o anche dette "puntuali", tra cui le operazioni aritmetiche (+, -, *, /), le operazioni di confronto (==, >, !=) e le operazioni logiche (&, |). Notate che il simbolo * e il simbolo / eseguono rispettivamente la moltiplicazione puntuale a le divisione puntuale, mentre per la moltiplicazione matriciale si utilizza il simbolo @.
Applicando l'operatore di somma (+) a due liste e poi a due array, noterete che il risultato è differente, provate a capire il motivo. Oltre alle operazioni, esistono moltissime funzioni e metodi che permettono di estrarre utili informazioni dagli array, di seguito se ne mostrano alcune:

`A.dtype`
: restituisce il `dtype` dell'array, cioè il tipo degli elementi dell'array.

`A.shape`
: restituisce una tupla contenente il numero di elementi lungo ogni dimensione.

`A.ndim`
: restituisce il numero di dimensioni dell'array.

`A.size`
: restituisce il numero totale di elementi dell'array.

`len(A)`
: restituisce il numero di elementi lungo la prima dimensione.

`A[:,::-1]`
: restituisce una matrice con l'ordine delle colonne invertito; `A[::-1,:]` opera in modo analogo sulle righe.

`np.cos(A)`
: restituisce una matrice con le stesse dimensioni di A contenente il coseno di ogni elemento di `A`, opera quindi punto per punto. Questo comportamento vale per numerose funzioni matematiche: `np.abs( )`, `np.log( )`, `np.exp( )`.

`np.min(A)`
: restituisce il valore minimo di tutti gli elementi dell'array; analogamente `np.max(A)` restituisce il massimo di tutti gli elementi e `np.mean(A)` restituisce le media di tutti gli elementi.

`np.min(A,0)`
: calcola il minimo lungo ogni colonna, mentre `np.min(A,1)` calcola il minimo lungo ogni riga. Anche `np.max(A,d)` e `np.mean(A,d)` funzionano in maniera analoga.

`A.flatten(order='C')`
: fornisce un vettore contenente i valori della matrice A costituito concatenando le righe. Analogamente `A.flatten(order='F')` fornisce un vettore contenente i valori della matrice A costituito concatenando le colonne.

Per salvare un array in un file, basta digitare il comando

```
np.save('dati.npy', A)
```

che crea nella directory corrente il file `dati.npy` che contiene i dati dell'array A. Gli array salvati possono essere caricati in memoria con il comando

```
A = np.load('dati.npy')
```

N.B. Numpy memorizza usualmente le matrici "per riga", ma è possibile memorizzarle anche "per colonna". E' preferibile usare sempre il formato "per riga" in quanto alcune funzioni sono applicabili solo a quest'ultimo formato. Nel caso in cui sia necessario convertire una matrice memorizzata "per colonna" nel formato "per riga", si può utilizzare la funzione `np.ascontiguousarray( )`.

6 Grafici usando Matplotlib

Per disegnare grafici in Python abbiamo bisogno di utilizzare la libreria Matplotlib. Prima di utilizzarla dalla console è necessario abilitare la modalità interattiva usando il *magic command* specifico:

```
In  [ ]: %matplotlib qt
```

Adesso possiamo importare il modulo `matplotlib.pyplot` che fornisce un insieme di funzioni per creare grafici con la sintassi simile a quella di MATLAB.

```
import matplotlib.pyplot as plt
```

Cominciamo col rappresentare una sequenza di punti definita mediante un vettore per i valori ed un vettore per l'asse dei tempi.

```
x = np.array([0, 0.3, 0.5, 0.8, 1, 0.8, 0.5, 0.3, 0])
t = np.arange(1, 5, 0.5)
```

Di seguito sono elencate alcune delle funzioni base di Matplotlib:

`plt.figure()`
crea una finestra.

`plt.stem(t,x)`
disegna la sequenza di valori di `x` come impulsi centrati negli istanti definiti dal vettore `t`.

`plt.plot(t,x)`
disegna il grafico interpolando linearmente i valori contenuti in `x`.

`plt.subplot(m,n,p)`
consente di visualizzare più grafici all'interno della stessa figura. Suddivide la figura in `m x n` quadranti e inserisce il grafico corrente nella posizione p-esima.

`plt.title('Titolo della figura')`
permette di inserire un titolo alla figura.

`plt.show()`
rende visibili le figure.

`plt.close()`
chiude la figura attiva.

Maggiori informazioni sulla libreria Matplotlib si trovano sul sito ufficiale `http://matplotlib.org`.

7 Alcuni suggerimenti generali

1. State sempre attenti all'indentazione, ricordatevi che un codice non indentato è sbagliato.
2. Scrivete funzioni brevi, ma usate nomi significativi, anche se lunghi, per le funzioni e le variabili.
3. Ricordare che l'indicizzazione inizia da 0 in Python.
4. Attenzione anche all'uso delle lettere maiuscole e minuscole (per Python sono diverse).
5. Create un codice ben commentato, scoprirete presto quanto sia utile avere una buona documentazione dei programmi che si vogliono usare.
6. Usate sempre l'help in linea quando avete dei dubbi su come deve essere usato un determinato comando.
7. Infine, fate molta attenzione ai messaggi di errore. Leggeteli sempre perché non solo vi danno informazioni sul tipo di errore, ma vi indicano anche il punto nel codice in cui il programma si blocca, e quindi dove verosimilmente è localizzato l'errore.

8 Esercizi proposti

a) Scrivete una funzione che analizza un vettore di numeri e restituisce un vettore saturando i valori superiori in modulo ad un dato limite. La firma della funzione deve essere la seguente: `clip(x, limit)`, dove `x` è il vettore in ingresso, `limit` è il valore limite, inoltre la funzione dovrà restituire un altro vettore. Per verificare la correttezza della funzione, applicatela al vettore `[-10, 3, -6, 0, 1, -2, 3, 4, -15, 3, 21]` usando il valore limite pari a `8`.

b) Scrivete una funzione in cui, data una matrice di valori positivi, reali e distinti tra loro, e dimensioni 16×16, volete conservare solo i 4 valori più grandi e annullare tutti gli altri. Il prototipo della funzione deve essere il seguente: `clip(x)`, dove `x` è la matrice in ingresso.

Suggerimento. Usate la funzione `sort` di numpy che realizza l'ordinamento.