



INTRODUZIONE A PYTHON

6 marzo 2019

COS'È PYTHON?



PYTHON è un linguaggio:

- **Ad alto livello** sintatticamente/semanticamente molto diverso da ling. macchina
- **Interpretato** eseguito linea dopo linea o tradotto in rappresentazione intermedia
- **Multi-uso** (scripting, web, GUI, scientific computing...)
- **Multi-paradigma** object oriented, imperative
- **Dynamically typed**
 - Uso diretto della variabile
 - Controllo tipo a runtime



PERCHÈ IMPARARLO?

- **Ampio bacino di utenti**

- **Aiuto online**
- **Librerie**
- Raspberry Pi



- Sostituisce gli pseudocodici
 - nei paper per descrivere algoritmi
- **Prototipazione rapida**
- **Multi piattaforma**

anaconda / packages

Packages Files Install Instructions

Filters
Type: all Access: all Label: all

Package Name	Access	Summary	Updated
vs2013_runtime	public	No Summary	2019-02-16
plac	public	The smartest command line arguments parser in the world	2019-02-16
gdata	public	No Summary	2019-02-16
pathlib	public	object-oriented filesystem paths	2019-02-16

« Previous showing 1 of 34 Next »

reddit r/learnpython Search r/learnpython

r/learnpython Posts wiki FAQ

Join the discussion BECOME A REDDITOR

Posted by u/justineff 2 years ago

ValueError: could not convert string to float

I am trying to write a program to calculate standard deviation and this is what I have so far:

```
import math
```

stackoverflow Search...

Home PUBLIC Stack Overflow Tags Users Jobs

Python: tile() returns weird error

I would like to do the following:

```
1 vectors = hstack((array([[nA, nM, nE]]), nP.T, (array([[nM, nA]])))  
tile(P[newaxis, newaxis, newaxis, ..., newaxis, newaxis], vectors)
```



Copyright (c) 1995-2001 Corporation for National Research Initiatives. All Rights Reserved.

Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam. All Rights Reserved.

Python for S60 is Copyright (c) 2004-2007 Nokia.

Version 1.4.0 final

Options Exit

DOWNLOAD PYTHON

- "Python 2 o Python 3"?
- **Distribuzioni Python**
 - Ecosistemi di librerie
- Esempi:
 - Cpython
 - ActivePython
 - Pypy
 - Jython
 - IronPython
 - WinPython
 - **Anaconda Python**
 - Pensato per scientific computing

Python 3.7 version

Download

64-Bit Graphical Installer (652.7 MB)
64-Bit Command Line Installer (557 MB)

Python 2.7 version

Download

64-Bit Graphical Installer (640.7 MB)
64-Bit Command Line Installer (547 MB)



INTERAGIRE CON PYTHON

- Interprete interattivo
 - Come la console di Matlab o R
 - (non è un concetto nuovo)
- Scripting
 - Codice scritto in un file di testo con estensione .py
- Quaderno interattivo
 - Jupyter notebook

```
(base) francesco@francesco-X556UB:~$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits()" or "quit()" for more
>>> print('Ciao mondo')
Ciao mondo
>>> 2+3
5
>>> help(print)
```

```
**** COMMODORE 64 BASIC U2 ****
64K RAM SYSTEM 38911 BASIC BYTES FREE
READY.
PRINT SQR(2+2)
2
READY.
PRINT LOG(1)
0
READY.
PRINT 2+3
8
READY.
PRINT 0.05-0.07+0.02+0
1.09139364E-11
READY.
```

$$H(z) = \prod_{\mu=1}^S \frac{b_{0,\mu} + b_{1,\mu} z^{-1} + b_{2,\mu} z^{-2}}{1 + a_{1,\mu} z^{-1} + a_{2,\mu} z^{-2}}$$

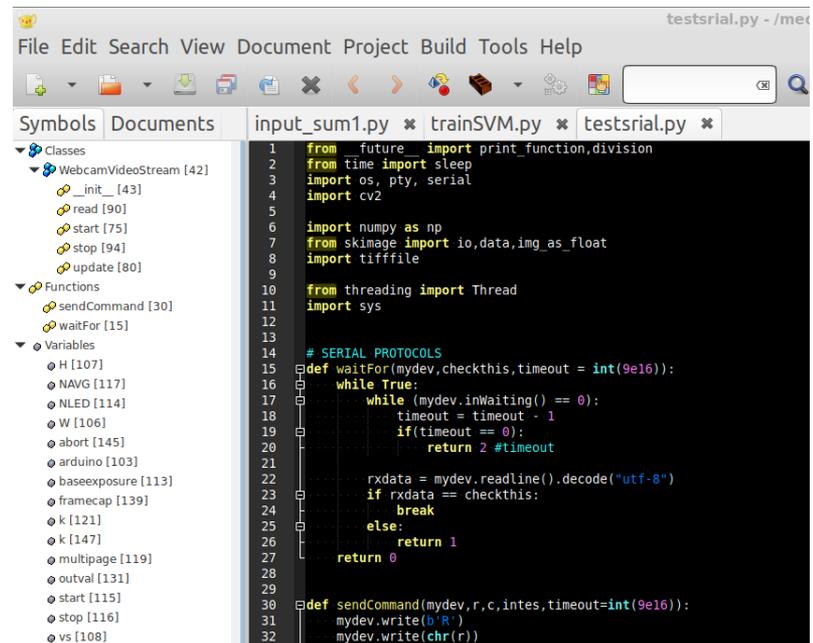
where $S = \lceil \frac{N}{2} \rceil$ denotes the total number of SOSs. These results state that any real valued system of order $N > 2$ can be decomposed into SOSs. This has a number of benefits

- quantization effects can be reduced by sensible grouping of poles/zeros, e.g. such that the spanned amplitude range of the filter coefficients is limited
- A SOS may be extended by a gain factor to further reduce quantization effects by normalization of the coefficients
- efficient and numerically stable SOSs serve as generic building blocks for higher-order recursive filters

Example - Cascaded second-order section realization of a lowpass

The following example illustrates the decomposition of a higher-order recursive Butterworth lowpass filter into a cascade of second-order sections.

```
1 [1]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.markers import MarkerStyle
from matplotlib.patches import Circle
import scipy.signal as sig
```

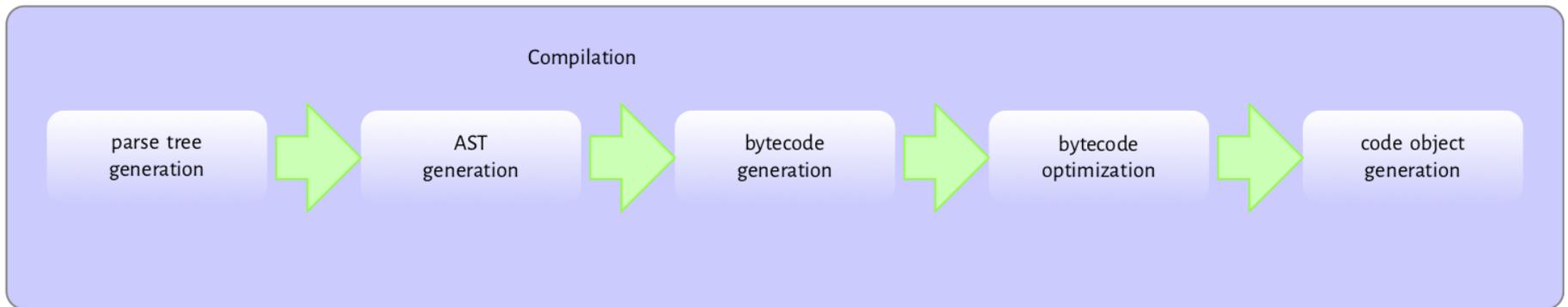


```
File Edit Search View Document Project Build Tools Help
input_sum1.py * trainSVM.py * testserial.py *
Symbols Documents
Classes
  WebcamVideoStream [42]
    __init__ [43]
    read [90]
    start [75]
    stop [94]
    update [80]
Functions
  sendCommand [30]
  waitFor [15]
Variables
  H [107]
  NAVG [117]
  NLED [114]
  W [106]
  abort [145]
  arduino [103]
  baseexposure [113]
  framecap [139]
  k [121]
  k [147]
  multipage [119]
  outval [131]
  start [115]
  stop [116]
  vs [108]
1  from future import print_function,division
2  from time import sleep
3  import os, pty, serial
4  import cv2
5
6  import numpy as np
7  from skimage import io,data,img_as_float
8  import tifffile
9
10 from threading import Thread
11 import sys
12
13
14 # SERIAL PROTOCOLS
15 def waitFor(mydev,checkthis,timeout= int(9e16)):
16     while True:
17         while (mydev.inWaiting() == 0):
18             timeout = timeout - 1
19             if(timeout == 0):
20                 return 2 #timeout
21
22         rxdata = mydev.readline().decode("utf-8")
23         if rxdata == checkthis:
24             break
25         else:
26             return 1
27     return 0
28
29
30 def sendCommand(mydev,r,c,intes,timeout=int(9e16)):
31     mydev.write(b'R')
32     mydev.write(chr(r))
```

MODELLO DI ESECUZIONE

```
python3 mioprogramma.py
```

- L'interprete legge il file da cima a fondo
 - Eventualmente segnala errori di sintassi
- Traduzione del sorgente (.py) in bytecode (.pyc)
 - Se il file non cambia, viene letto direttamente il pyc
- Python Virtual Machine



```
def f(x, y):  
    return x + y
```



2	0	LOAD_FAST	0	(x)
3		LOAD_FAST	1	(y)
6		BINARY_ADD		
7		RETURN_VALUE		

SINTASSI ELEMENTARE



PRIMO PROGRAMMA

```
"""Questo è
    un
    commento"""
print('Ciao a titti')
"""fine"""
#exit()
# possiamo omettere quest'istruzione
```

ELEMENTI:

- commento multiriga introdotto da `"""` . . . `"""`
- funzione `print` che stampa sullo schermo una costante stringa
- funzione `exit()` che chiude il programma
- un commento inline

INPUT DA TASTIERA

```
10 print('Somma due numeri')
11 num1 = input('enter number 1: ')
12 num2 = input('enter number 2: ')
13 out = num1 + num2
14 print('out', out)
15 print('end')
16
```

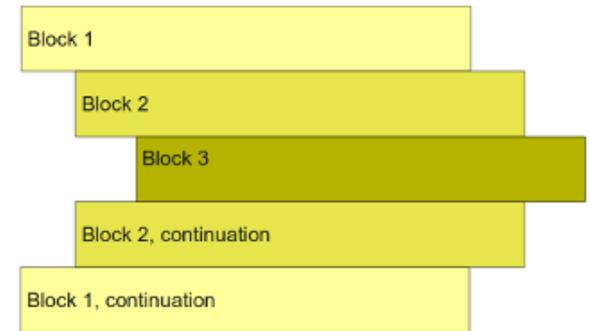
Somma di due numeri inseriti da tastiera

- Assegnamento con =
- Nessuna dichiarazione di variabile
- Funziona?
- E se non metto numeri?

BLOCCHI DI CODICE

- Spazi e fine linea SINTATTICAMENTE RILEVANTI
 - Identificano e delimitano blocchi di istruzioni
- Istruzioni allo stesso livello di indentazione appartengono allo stesso blocco
- Forza a scrivere codice leggibile e autoriferenziale

```
if y:  
    if z:  
        f1()  
else:  
    f2()
```



```
9  #include <stdio.h>  
10  
11  int main()  
12  {  
13      int a = 2;  
14      if (a > 2)  
15      {  
16          if(a%2 == 0){  
17              printf("pari");  
18          }  
19      }  
20      else  
21          printf("minore di 2");  
22  }
```

STILE DEL CODICE

- Guida ufficiale (PEP8)
 - www.python.org/dev/peps/pep-0008
- Linee generali:
 - 4 spazi al posto del tab \t
 - 1 istruzione per linea, massimo 79 caratteri
 - Istruzioni lunghe possono essere disposte su più linee con \
 - Nessun terminatore di linea (;)
 - Omissione delle () attorno alle condizioni
 - Nomi delle classi in CamelCase
 - Il resto in `in_minuscolo_con_underscore`

VARIABILI E ASSEGNAMENTI

- **Dichiarazione** e specificazione del **tipo** non necessarie
- **Assegnamenti** con =
- **Shortcuts** come in C
 - +=, -=
 - *=, /=
 - %=
- Più assegnamenti per riga
 - **1-line swap!**

```
>>> a,b = 3+4, 'pippo'
>>> a
7
>>> b
'pippo'
```

```
>>> a,b = 7,9
>>> print(a,b)
7 9
>>> a,b = b,a
>>> print(a,b)
9 7
```

```
18 LOAD_NAME          0 (a)
20 LOAD_NAME          1 (b)
22 ROT_TWO
24 STORE_NAME         1 (b)
26 STORE_NAME         0 (a)
```

BUILT-IN DATA TYPES



- **Tipi base:**
 - **NoneType:** *None*
 - **Numerics:** `int(50)`, `long(12345678L)`, `float(5.5)`
 - **Bool:** *True*, *False*
- **Tipi contenitore:**
 - **str:** `'Les Paul'`, `"Stratocaster"`
 - **list:** `[1, 2, 3]` (elementi eterogenei `[1, 2.0, 3, "stella"]`)
 - **tuple:** `(1, 2, 3)`
 - **dict:** `auto = {'Nome': 'Xsara', 'coppia': 13.8}`
 - **set:** `{1, 2, 2, 3} → set([1, 2, 3])`



OPERATORI NUMERICI

- Operatori unari:
 - `+`, `-`, `not` (anteposti al numero)
- Operatori binari
 - `+`, `-`, `*`, `/`, `//`, `%`, `**`
 - Divisione float `/`
 - Divisione intera `//`
- Type casting
 - `int()`, `float()`
- Modulo `math` della Python std lib



STRINGHE



- **Sequenza ordinata (indicizzabile)** di caratteri compresa tra apici singoli, doppi o tripli

```
Stringa = 'prova'
```

```
A = Stringa[0]; B = Stringa[2]
```

- # A contiene 'p', B contiene 'o'
- # indice parte da 0

- **Tipo immutabile**

- Non posso fare l'inverso! Produce un errore
- `Stringa[3] = 't'`

```
>>> b = 'Ibanez Jem'
>>> b[3] = 't'
Traceback (most recent call last):
  File "<pyshell#104>", line 1, in <module>
    b[3] = 't'
TypeError: 'str' object does not support item assignment
>>>
```

STRINGHE

- Ricerca elementi (membership testing)

```
>>> stringa = "Oggi e' il 6 marzo"
>>> print(stringa)
Oggi e' il 6 marzo
>>> pattern = 'g'
>>> pattern in stringa
True
>>> sub_patrn = 'mar'
>>> sub_patrn in stringa
True
```

- Possiamo migliorare l'uscita?
 - Controllo programmatico?

Istruzioni condizionali: IF/ELSE

- Costrutto usato per prendere decisioni

```
If condizione :                due punti!  
    istruzione 1                4 spazi indentazione  
    istruzione 2  
else :                          due punti!  
    Istruzione 3                4 spazi indentazione  
    Istruzione 4
```

- *Condizione* è tutto ciò che restituisce **boolean**
- Non c'è switch (cascata di **elif**)

OPERATORI DI CONFRONTO

- `conf_op = ['<', '<=', '==', '!=', '>=', '>']`
 - A `conf_op` B equivale a **True** o **False**
- Possono essere combinati con gli operatori logici
 - `Log_op = ['and', 'or', 'not']`
 - Che a loro volta possono essere combinati tra di loro
- **UGUAGLIANZA**
 - ==** deep equality (tra valori)
 - Funziona per tutti gli oggetti, per oggetti di tipo differente ritorna False
 - Is** shallow equality (per oggetti)
 - `A is B` ritorna True se A e B SONO lo stesso oggetto

OPERATORI DI CONFRONTO

Similmente al C:

- `bool(0) -> False`

- `bool(1) -> True`

- `bool(None) -> False`

- `bool(contenitore vuoto) -> False`

- `bool(qualsiasi) -> True`

STRINGHE

```
stringa = input('Stringa raw: ')\npattern = input('pattern: ')\nif pattern in stringa:\n    print("c'è")\nelse:\n    print("non c'è")\nprint('end')
```

Miglioriamo il riuso del codice

FUNZIONI

- Unità funzionali per il riuso di codice (spesso più utili dei programmi!)
- Sintassi per la **definizione**: molto semplice!!

```
def nome_fun (argomenti) :      due punti!  
    istruzione1                 4 spazi indentazione  
    Istruzione2
```

- Sintassi per **richiamarla**:

```
nome_fun (argomenti)
```

STRINGHE E FUNZIONI

```
def check_string(stringa, pattern): # funzione ritorna valori
    if pattern in stringa:
        contenuta = True
    else:
        contenuta = False
    quante_volte = stringa.count(pattern)
    return (contenuta,quante_volte) # una tupla
```

```
def print_bool(bool_val): # procedura, non ritorna niente
    if bool_val:
        print("C'è")
    else:
        print('Manca')
```

```
## codice principale
s = input('Stringa: ')
p = input('cosa vuoi cercare: ')
out = check_string(s,p)
print(type(out), out[0],out[1])
print_bool(out[0])
print('end')
```

FUNZIONI RICORSIVE

Una funzione ricorsiva è una funzione ricorsiva

```
def conto_alla_rovescia(num_part):  
    if n<=0:  
        print('Via!') # condizione di stop  
    else:  
        print(num_part)  
        conto_alla_rovescia(num_part-1)
```

- Condizione di stop: ESSENZIALE
 - *"Che al mercato mio padre comprò"*
- Alto costo computazionale
- Limite ricorsioni



RecursionError: maximum recursion depth exceeded while pickling an object



FUNZIONI RICORSIVE

- Esempio classico: **Fattoriale**

$0! = 1$ # condizione di stop

$n! = n(n-1)!$

Ma per num float??

Implementazione non funziona

```
def fattoriale(num):  
    print('ric:', num)  
    if num == 0:  
        return 1  
    else:  
        return num*fattoriale(num-1)
```

Correggiamo:

- Aggiunta **condizione di guardia**

- Spesso sono più lunghe della funzione

```
def fattoriale(num):  
    if not isinstance(num, int):  
        print('Non è una gamma')  
        return None  
    print('ric:', num)  
    if num <= 1: # num == 0  
        return 1  
    else:  
        return num*fattoriale(num-1)
```

CICLI

- Strutture che eseguono più volte un blocco di codice
 - Numero di iterazioni noto: **for** loop
 - Numero di iterazioni non noto: **while** loop

- WHILE

while **condizione**(variabile):

Istruzione 1

Istruzione ... n

Modifica variabile

```
def conto_rovescia_while(num):
```

```
    while num > 0:
```

```
        print(num)
```

```
        num -= 1
```

```
    print('Via!')
```

- Radici quadrate babilonesi

$$y = \sqrt{a} \approx \sum \frac{x + \frac{a}{x}}{2}$$

CICLI: radice babilonese

```
def radice_quadr_bab(num, old):
```

```
    i = 0
```

```
    while True:
```

```
        print(i)
```

```
        y = 0.5*(old + num/old)
```

```
        if y == old:
```

```
            break
```

```
        old = y
```

```
        i += 1
```

```
    return y
```

```
def radice_quadr_migl(num, old=None, epsilon=1e-10):
```

```
    i = 0
```

```
    if old == None:
```

```
        old = num
```

```
    while True:
```

```
        y = 0.5*(old + num/old)
```

```
        if abs(y-old) < epsilon:
```

```
            break
```

```
        old = y
```

```
        i += 1
```

```
    return y, i
```

CICLO FOR

```
for variabile in container:  
    istruzione1  
    Istruzione2
```

- Non è un "vero" ciclo for, ma un for each!
- **Variabile** assume di volta in volta il valore di uno degli elementi di **lista**, **sequenzialmente**
- L'assegnazione è gestita internamente dal costrutto

LISTE DI NUMERI

```
lista_num = range(inizio, fine, step) # python2  
lista_num = list(range(inizio, fine, step)) # python3
```

```
Lista_num = list(range(0,10,3))  
              [0,3,6,9]  
len(lista_num) --> 4
```

```
for idx, el in enumerate(['c','i','a','o']):  
    print(idx,el)
```

Generazione lista:

- [Parte da Inizio compreso] **opz**
- Termina a Fine escluso
- [Step: passo per nuovo elemento] **opz**

ESEMPIO: CRITTOGRAFIA AVANZATA

- Alfabeto farfallino:
 - Ad ogni vocale si fa prefazione fonetica una vocale e (d'altronde è un messaggio segreto)
 - Mapping di vocali: [a,e,i,o,u] --> [afa,efe,ifi,ofu,ufu]
 - Consonanti inalterate
- Costruiamo due funzioni:
 - Cripta da stringa
 - Decripta da stringa
 - Usiamole in un programma che gestisce I/O

ESEMPIO CRITTOGRAFIA

```
# funzioni
def cripta_stringa(stringa):
    messaggio_criptato = "" # stringa vuota
    stringa = stringa.lower()
    for car in stringa:
        if car in "aeiou":
            messaggio_criptato += car + 'f' + car
        else:
            messaggio_criptato += car
    return messaggio_criptato

def decripta_stringa(stringa):
    #messaggio = stringa
    for codice in ("afa", "efe", "ifi", "ofu", "ufu"): # tupla di stringhe
        vocale = codice[0] # indirizzamento stringa
        stringa = stringa.replace(codice, vocale)
        #print(codice,messaggio)
    return stringa

# start
raw_string = 'che bella giornata'
print('raw', raw_string)
cript_string = cripta_stringa(raw_string)
print('cript', cript_string)
decr_string = decripta_stringa(cript_string)
print('decode', decr_string)
```

LISTE

- Inizializzazione vuota

```
lista = []
```

- Inizializzazione esplicita

```
lista = [1, 2, 3, 10, 190]
```

- Append consecutivi

```
Lista = []
```

```
for ii in range(10):  
    Lista.append(ii)
```

LISTE

- Generiamo una lista dei quadrati di una lista

```
lista = list(range(15))
newlista = []
for el in lista:
    if el%2:
        newlista.append(el**2)
print(newlista)
```

Metodo più veloce: **LIST Comprehensions**

```
new2 = [ el**2 for el in lista ]
print(new2)
```

```
new3 = [el **2 for el in lista if el%2]
print(new3)
```