Εφαρμογές Η/Υ

5ο Μάθημα

- Διαχείριση λαθών εκτέλεσης. Εξαιρέσεις
- Δημιουργία και χειρισμός αριθμητικών δεδομένων
  - Μητρώα
  - Γραφήματα

Ν. Λαγαρός, Θ. Στάμος, Χ. Φραγκουδάκης

Εισαγωγή στην Python

Λάθη κατά τη διάρκεια της εκτέλεσης

- Λάθη κατά τη διάρκεια της εκτέλεσης δημιουργούν εξαιρέσεις (exceptions)
- Διαφορετικά λάθη δημιουργούν διαφορετικές εξαιρέσεις

```
>>> 1 / 0
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
>>> 1 + 'e'
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
>>> d = {1:1, 2:2}
>>> d[3]
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 3
```

Εισαγωγή στην Python

Όλα τα ονόματα των εξαιρέσεων: `Exception`, `StopIteration`, `SystemExit`, `StandardError`, `ArithmeticError`, `OverflowError`, `FloatingPointError`, `ZeroDivisonError`, `AssertionError`, `AttributeError`, `EOFError`, `ImportError`, `KeyboardInterrupt`, `LookupError`, `IndexError`, `KeyError`, `NameError`, `UnboundLocalError`, `EnvironmentError`, `IOError`, `OSError`, `SyntaxError`, `IndentationError`, `SystemError`, `SystemExit`, `TypeError`, `ValueError`, `RuntimeError`, `NotImplementedError`

```
>>> l=[1, 2 , 3]
>>> l[4]
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

```
>>> l.lastElement
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'list' object has no attribute 'lastElement'
```

Εισαγωγή στην Python

Διαχείριση εξαιρέσεων με χρήση των `try` / `except`

```
while True:
    try:
        x = int(input('Please enter a number: '))
        break
    except ValueError:
        print('That was no valid number. Try again...')
```

```
Please enter a number: a
That was no valid number. Try again...
Please enter a number: 1
>>> x
1
```

Τι συνέβη μέσα στο `try` / `except`:

```
>>> int('a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'a'
```

Εισαγωγή στην Python

Διαχείριση εξαιρέσεων με χρήση των `try` / `except` / `else`

```
files = ['file1.txt', 'file2.txt', 'file3.txt']

for file in files:
    try:
        f = open(file, 'r')
    except IOError:
        print 'cannot open', file
    else:
        print file, 'has', len(f.readlines()), 'lines'
        f.close()
```

- Το "σώμα" του `try` / `except` πρέπει να είναι όσο το δυνατό μικρότερο
- Αποφεύγουμε έτσι άλλο ενδεχόμενο εκτός του `IOError`
- Αν δεν συμβεί εξαίρεση εκτελείται το σώμα του `else`

Εισαγωγή στην Python

Διαχείριση εξαιρέσεων με χρήση των `try` / `except` / `else` / `finally`

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:                    # Διαίρεση με το μηδέν
        print "division by zero!"
    else:                              # Εκτελείται αν δεν υπάρξει κάποια εξαίρεση
        print "result is", result
    finally:   # Εκτελείται οπωσδήποτε στο τέλος ανεξάρτητα των εξαιρέσεων
        print "executing finally clause"
```

```
>>> divide(2, 1)
result is 2
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Εισαγωγή στην Python

Πολλαπλά `except`

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as e:
    print "I/O error({0}): {1}".format(e.errno, e.strerror)
except ValueError:
    print "Could not convert data to an integer."
except:
    print "Unexpected error:", sys.exc_info()[0]
    raise
```

- Χειρισμός κάθε εξαίρεσης ξεχωριστά
- Αν δεν συμβεί κάποια από τις γνωστές εξαιρέσεις εισάγουμε μία με το `raise`

Εισαγωγή στην Python

Χρήση των εξαιρέσεων για πέρασμα μυνημάτων μέσα στον κώδικα

```
def achilles_arrow(x):
    if abs(x-1) < 1e-3:
        raise StopIteration     # εισάγει την εξαίρεση StopIteration
    x = 1 - (1-x)/2
    return x

x = 0
while True:
    try:
        x = achilles_arrow(x)
    except StopIteration:       # ανιχνεύτηκε εξαίρεση StopIteration
        break                   # έξοδος από το while loop
```

```
>>> x
0.9990234375
```

- Η συνάρτηση περνά το μήνυμα `StopIteration` στο κυρίως πρόγραμμα

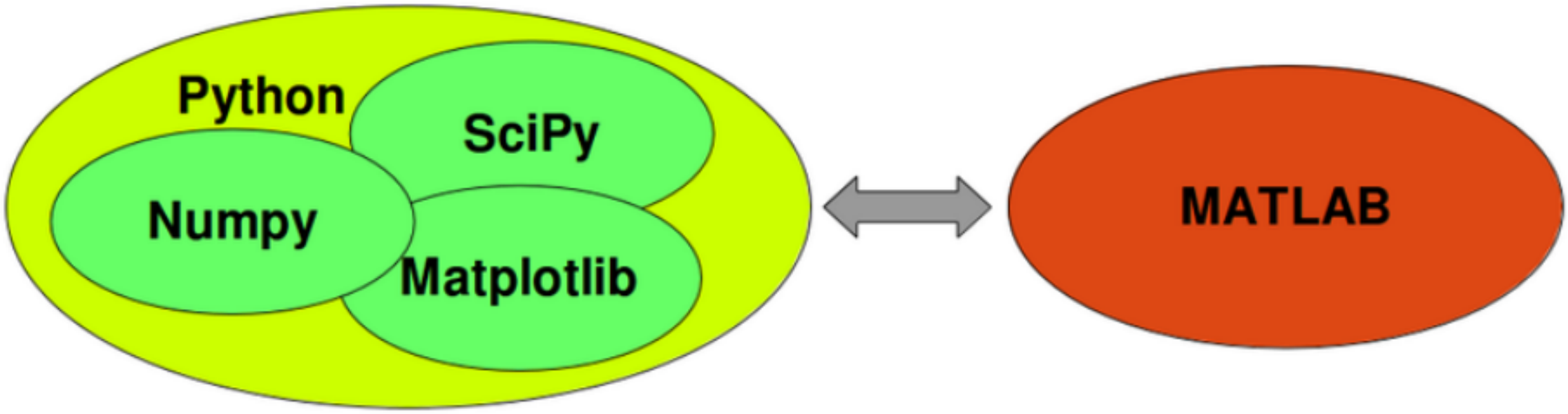
Εισαγωγή στην Python

Οι εξαιρέσεις είναι αντικείμενα

```
class Error(Exception):
    """Base class for exceptions in this module."""
    pass

class InputError(Error):
    """Exception raised for errors in the input.

    Attributes:
        expression -- input expression in which the error occurred
        message -- explanation of the error
    """

    def __init__(self, expression, message):
        self.expression = expression
        self.message = message
```

- Σύμβαση για τη δημιουργία νέων εξαιρέσεων εκτός των προκαθορισμένων

Η Python σαν ενναλακτική λύση αντί του Matlab



- `NumPy`: υπολογισμοί προσανατολισμένοι στα μητρώα
- `SciPy`: σπιτημονικοί υπολογισμοί
- `Matplotlib`: γραφήματα όπως το Matlab

Χρήση του `NumPy`

Απλό παράδειγμα:

```
import numpy as np
C = np.array([25.3, 24.8, 26.9, 23.9]) #  Βαθμοί Κελσίου
F = C * 9 / 5 + 32      #  Μετατροπή σε βαθμούς Φαρενάιτ
print(F)
```

```
[ 77.54  76.64  80.42  75.02]
```

- Ένα μονοδιάστατο μητρώο του `NumPy` αρχικοποιείται με μια λίστα
- Τα μητρώα του `NumPy` υποστηρίζουν βαθμωτό πολλαπλασιασμό

Αντίστοιχα με τη βασική Python:

```
cvalues = [25.3, 24.8, 26.9, 23.9]
fvalues = [ x*9/5 + 32 for x in cvalues]
print(fvalues)
```

```
[77.54, 76.64, 80.42, 75.02]
```

Χρήση του `NumPy`

Δημιουργία ομοιόμορφα κατανεμημένων τιμών με χρήση του `arange`:

```
import numpy as np
a = np.arange(1, 10)
print(a)
x = np.arange(10.4)
print(x)
x = np.arange(0.5, 10.4, 0.9)
print(x)
```

```
[1 2 3 4 5 6 7 8 9]
[  0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
[ 0.5  1.4  2.3  3.2  4.1  5.   5.9  6.8  7.7  8.6  9.5]
```

`arange([start,] stop[, step,], dtype=None)`

- `start`: αρχική τιμή (προκαθορισμένη τιμή το 0)
- `stop`: τέλος του διαστήματος (συνήθως δεν περιλαμβάνεται)
- `step`: η απόσταση μεταξύ των τιμών (προκαθρισμένη τιμή το 1)
- `dtype`: τύπος τιμών (συνήθως συνάγεται από τις άλλες παραμέτρους)

Χρήση του `NumPy`

Δημιουργία ομοιόμορφα κατανεμημένων τιμών με χρήση του `linspace`:

```
import numpy as np
print(np.linspace(1, 10))                    # 50 τιμές στο [1,10]
print(np.linspace(1, 10, 7))                 #  7 τιμές στο [1,10]
print(np.linspace(1, 10, 7, endpoint=False)) #  7 τιμές στο [1,10)
```

```
[  1.          1.18367347   1.36734694   1.55102041   1.73469388
   1.91836735   2.10204082   2.28571429   2.46938776   2.65306122
   2.83673469   3.02040816   3.20408163   3.3877551    3.57142857
   3.75510204   3.93877551   4.12244898   4.30612245   4.48979592
   4.67346939   4.85714286   5.04081633   5.2244898    5.40816327
   5.59183673   5.7755102    5.95918367   6.14285714   6.32653061
   6.51020408   6.69387755   6.87755102   7.06122449   7.24489796
   7.42857143   7.6122449    7.79591837   7.97959184   8.16326531
   8.34693878   8.53061224   8.71428571   8.89795918   9.08163265
   9.26530612   9.44897959   9.63265306   9.81632653  10.        ]
[  1.    2.5   4.    5.5   7.    8.5  10. ]
[ 1.         2.28571429  3.57142857  4.85714286  6.14285714  7.42857143
  8.71428571]
```

Χρήση του `NumPy`

Δημιουργία ομοιόμορφα κατανεμημένων τιμών με χρήση του `linspace`:

`linspace(start, stop, num=50, endpoint=True, retstep=False)`

- Επιστρέφει `num` τιμές ομοιόμορφα κατανεμημένες στο $[\text{start}, \text{stop}]$
- Αν `endpoint==False` τότε το διάστημα είναι $[\text{start}, \text{stop})$
- Αν `retstrep==True` επιστρέφει πλειάδα με 2ο στοιχείο το `step`

```
import numpy as np
samples, spacing = np.linspace(1, 10, retstep=True)
print(spacing)
samples, spacing = np.linspace(1, 10, 20, endpoint=True, retstep=True)
print(spacing)
samples, spacing = np.linspace(1, 10, 20, endpoint=False, retstep=True)
print(spacing)
```

```
0.1836734693877551
0.47368421052631576
0.45
```

Χρήση του `NumPy`

Τα μητρώα του `NumPy` είναι πιο αποδοτικά από τις λίστες:

```
import time, numpy as np
size_of_vec = 1000

def pure_python_version():
    t1 = time.time()
    X = range(size_of_vec)
    Y = range(size_of_vec)
    Z = []
    for i in range(len(X)):
        Z.append(X[i] + Y[i])
    return time.time() - t1

def numpy_version():
    t1 = time.time()
    X = np.arange(size_of_vec)
    Y = np.arange(size_of_vec)
    Z = X + Y
    return time.time() - t1

t1, t2 = pure_python_version(), numpy_version()
print("Numpy is in this example " + str(t1/t2) + " faster!")
```

```
Numpy is in this example 17.771058315334773 faster!
```

Χρήση του `NumPy`

Δημιουργία 0-διάστατου μητρώου:

```
import numpy as np
x = np.array(42)
print("x: ", x)
print("The type of x: ", type(x))
print("The dimension of x:", np.ndim(x))
```

```
x:  42
The type of x:  <class 'numpy.ndarray'>
The dimension of x: 0
```

- Οι αριθμοί είναι 0-διάστατα μητρώα για το `NumPy`
- Η κλάση των μητρώων είναι η `ndarray`
- `ndim(x)` επιστρέφει τη διάσταση του μητρώου `x`

Χρήση του `NumPy`

Δημιουργία 1-διάστατου μητρώου:

```
F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
V = np.array([3.4, 6.9, 99.8, 12.8])
print("F: ", F)
print("V: ", V)
print("Type of F: ", F.dtype)
print("Type of V: ", V.dtype)
print("Dimension of F: ", np.ndim(F))
print("Dimension of V: ", np.ndim(V))
```

```
F:  [ 1  1  2  3  5  8 13 21]
V:  [  3.4   6.9  99.8  12.8]
Type of F:  int64
Type of V:  float64
Dimension of F:  1
Dimension of V:  1
```

Σε αντίθεση με τις λίστες, τα μητρώα του `NumPy` μπορεί περιέχουν μόνο ίδιου τύπου δεδομένα που μπορεί να εξεταστούν/οριστούν με το `dtype`

Χρήση του `NumPy`

Δημιουργία 2-διάστατου μητρώου:

```
import numpy as np
A = np.array([[3.4, 8.7, 9.9],
              [1.1, -7.8, -0.7],
              [4.1, 12.3, 4.8]])
print(A)
print(A.ndim)
```

```
[[  3.4   8.7   9.9]
 [  1.1  -7.8  -0.7]
 [  4.1  12.3   4.8]]
2
```

Χρήση του `NumPy`

Δημιουργία 3-διάστατου μητρώου:

```
import numpy as np
B = np.array([ [[111, 112], [121, 122]],
               [[211, 212], [221, 222]],
               [[311, 312], [321, 322]] ])
print(B)
print(B.ndim)
```

```
[[[111 112]
  [121 122]]

 [[211 212]
  [221 222]]

 [[311 312]
  [321 322]]]
3
```

Χρήση του `NumPy`

Δημιουργία 4-διάστατου μητρώου:

```
import numpy as np
C = np.array([[[[1111, 1112], [1121, 1122]], [[1211, 1212], [1221, 1222]], [[1311,
print(C)
print(C.ndim)
```

```
[[[[ 1111  1112]
   [ 1121  1122]]

  [[ 1211  1212]
   [ 1221  1222]]

  [[ 1311  1312]
   [ 1321  1322]]]


 [[[ 2111 22112]
   [ 2121  2122]]

  [[ 2211  2212]
   [ 2221  2222]]

  [[ 2311  2312]
   [ 2321  2322]]]]
4
```

Χρήση του `NumPy`

Απλή αριθμητική μητρώων:

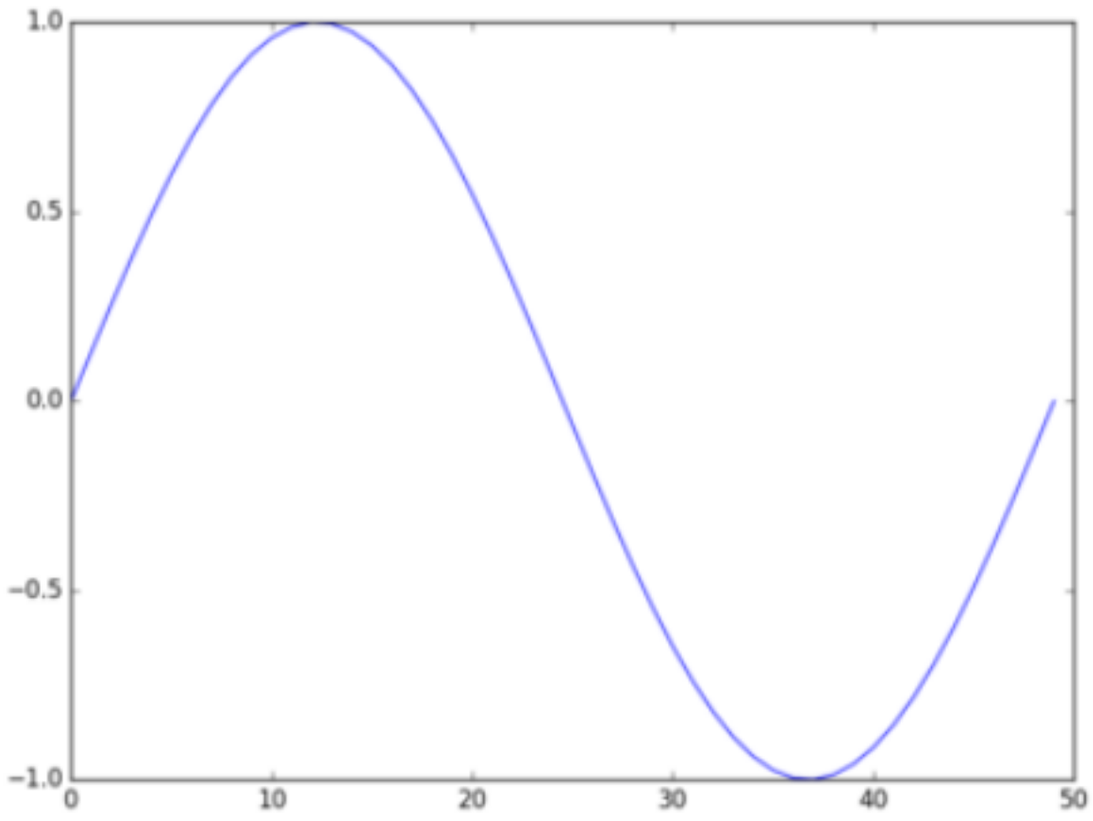
```
import numpy as np
a = np.array([1, 2, 3, 4])
b = np.array([2, 3, 4, 5])
print('a = ', a)
print('b = ', b)
print('a + b  =', a+b)
print('a * b  =', a*b)
print('a ** b =', a**b)
print(np.pi)
print(np.e)
```

```
a =  [1 2 3 4]
b =  [2 3 4 5]
a + b  = [3 5 7 9]
a * b  = [ 2  6 12 20]
a ** b = [   1    8   81 1024]
3.141592653589793
2.718281828459045
```

Χρήση του `NumPy` και του `Matplotlib`

Παραγωγή γραφημάτων από μητρώα:

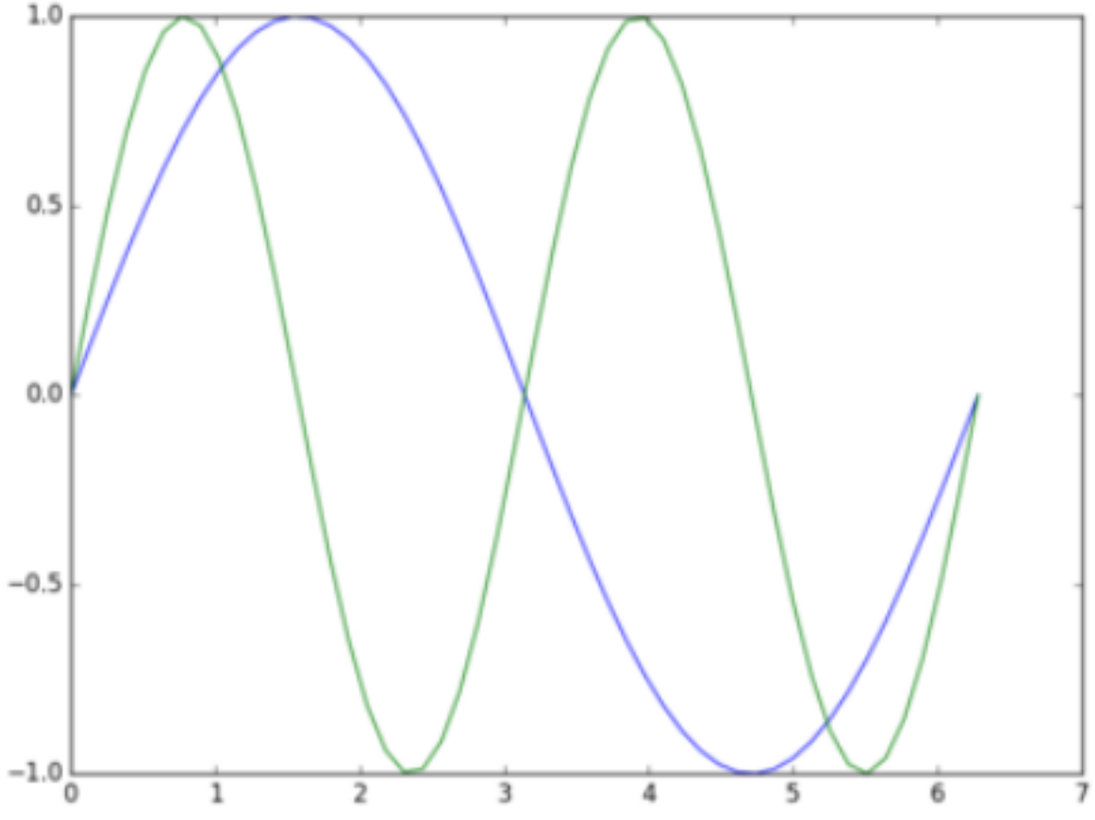
```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```



Χρήση του `NumPy` και του `Matplotlib`

Απεικόνιση πολλαπλών δεδομένων:

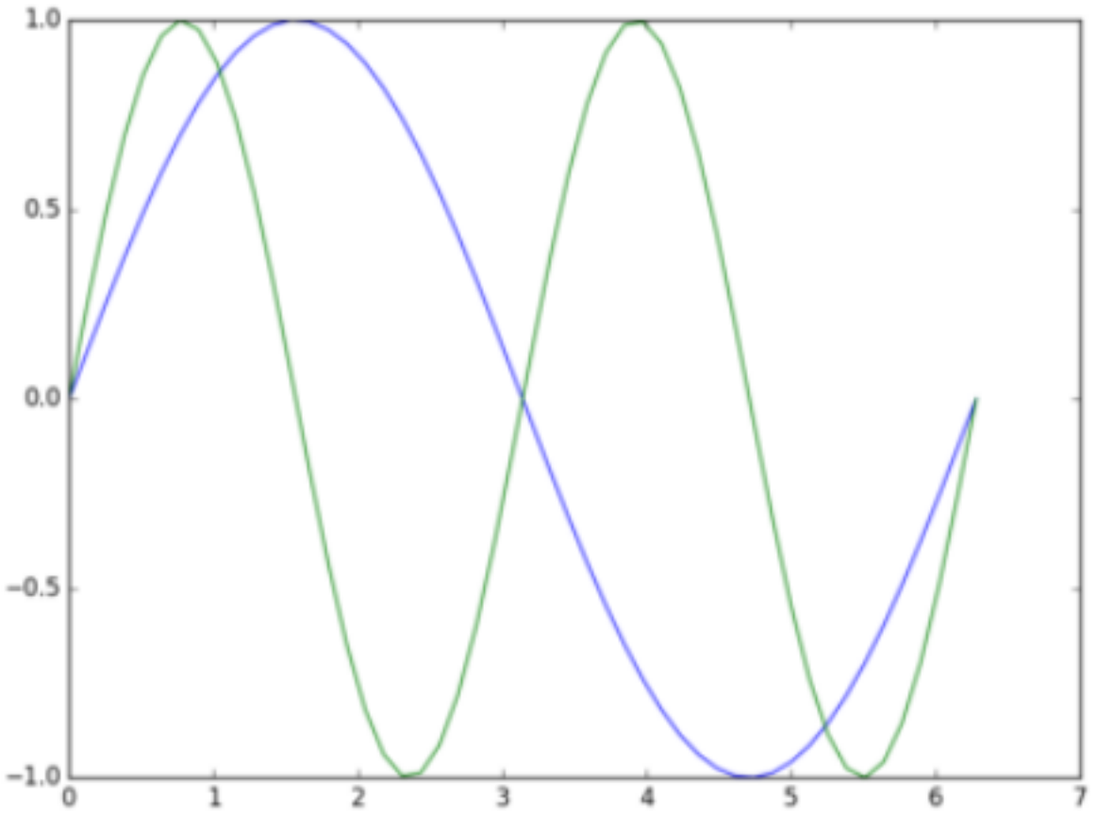
```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi)
y0 = np.sin(x)
y1 = np.sin(2*x)
plt.plot(x, y0, x, y1)
plt.show()
```



Χρήση του `NumPy` και του `Matplotlib`

Ενναλακτικά με τη χρήση του `pylab`:

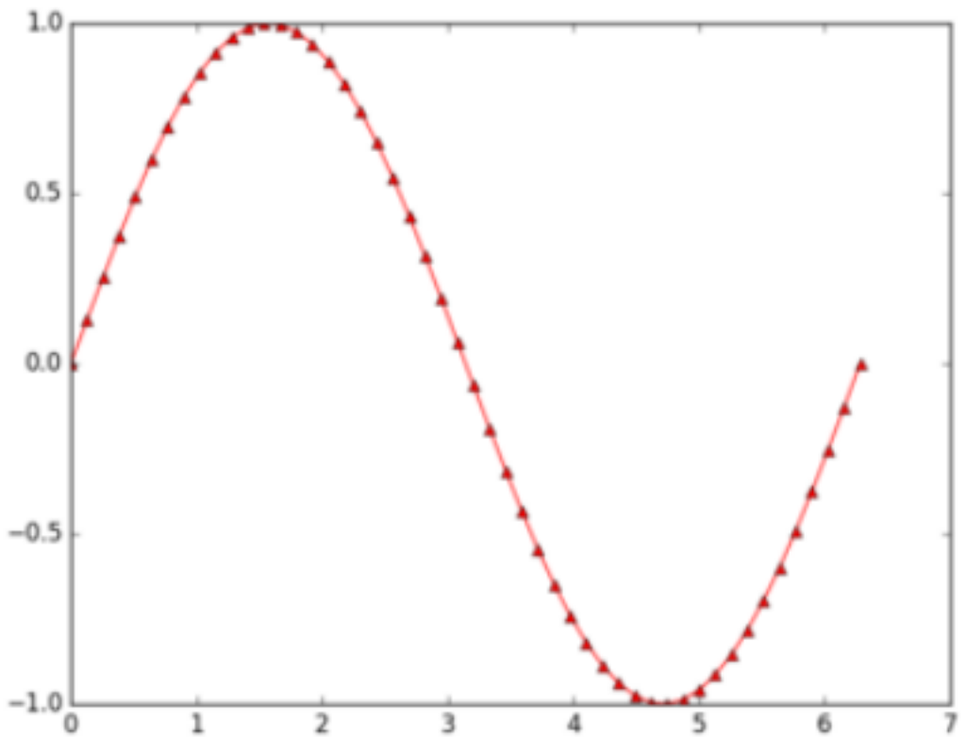
```
from pylab import * # imports numpy scipy and matplotlib globally
x = linspace(0, 2*np.pi)
y0 = sin(x)
y1 = sin(2*x)
plot(x, y0, x, y1)
show()
```



Χρήση του `NumPy` και του `Matplotlib`

Μορφοποίηση γραφημάτων:

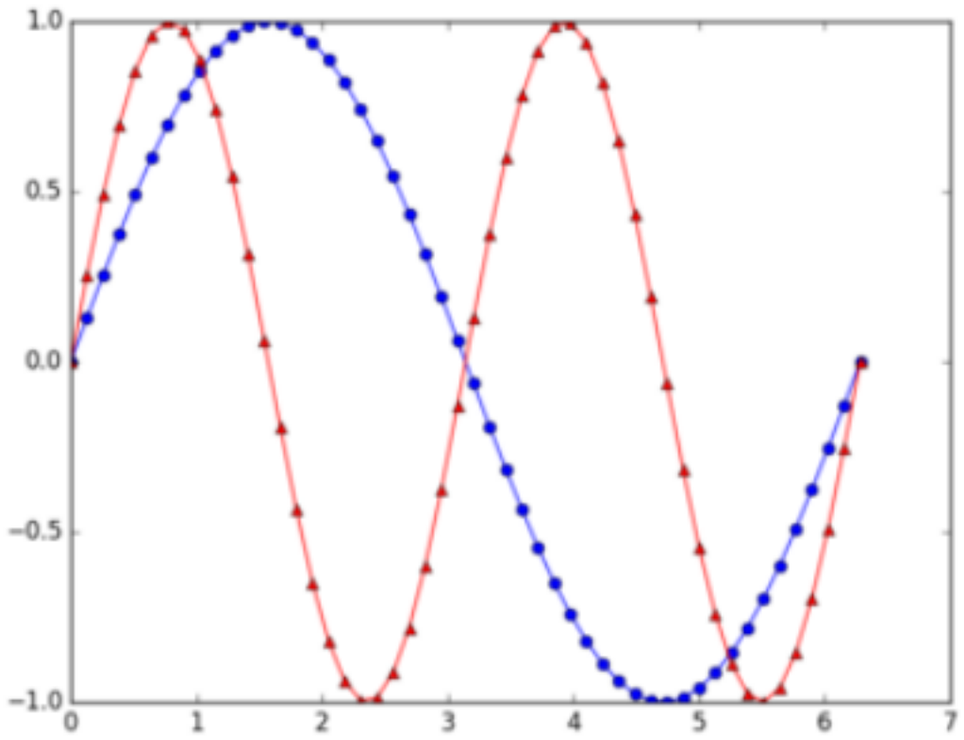
```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi)
y = np.sin(x)
plt.plot(x, y, 'r-^') # red, solid line, triangles as marks
plt.show()
```



Χρήση του `NumPy` και του `Matplotlib`

Μορφοποίηση γραφημάτων:

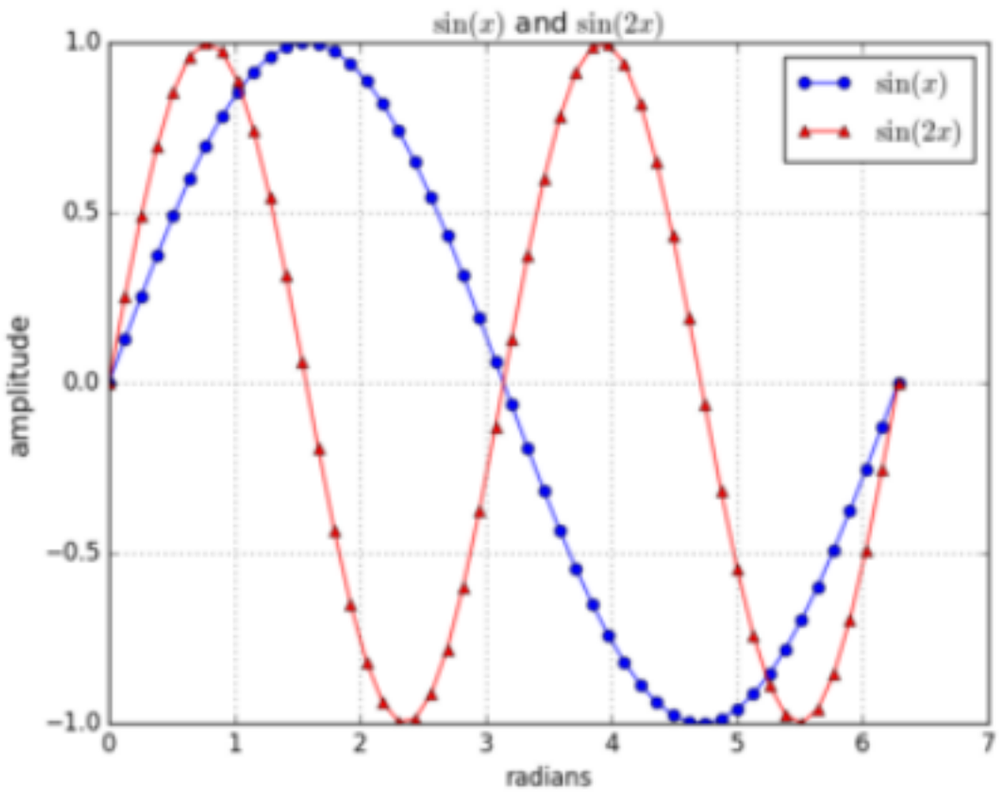
```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi)
y0 = np.sin(x)
y1 = np.sin(2*x)
plt.plot(x, y0, 'b-o', x, y1, 'r-^')
plt.show()
```



Χρήση του `NumPy` και του `Matplotlib`

Υπόμνημα, τίτλοι στους άξονες και στο γράφημα, κάνναβος

```
import numpy as np, matplotlib.pyplot as plt
x, y0, y1 = np.linspace(0, 2*np.pi), np.sin(x), np.sin(2*x)
plt.plot(x, y0, 'b-o', label='$\sin(x)$')
plt.plot(x, y1, 'r-^', label='$\sin(2x)$')
plt.xlabel('radians')
plt.ylabel('amplitude', fontsize='large')
plt.title('$\sin(x)$ and $\sin(2x)$')
plt.legend()
plt.grid()
plt.show()
```



Ερωτήσεις;