Python for Matlabbers

Using the Jupyter Notebook

See Help->Keyboard shortcuts. Some useful shortcuts are:

- Enter: enter edit mode
- ESC: exit edit mode
- Up-down keys in exit mode: move between cells
- Ctrl+Enter: Run cell
- Shift+Enter: Run cell and move to the next cell
- Alt+Enter: Run cell and insert below
- Tab: Used to auto-complete

In [3]:

```
#Ipython configuration options..
#tell Jupyter to show any plots within this page.
%matplotlib inline
```

Python Syntax

Comments, Assignment, Printing

In [2]:

```
# single-line comment
print('hello'); #endofline comment

"""
multiline
comment
"""
```

```
hello
```

Out[2]:

```
'\nmultiline\ncomment\n'
```

NOTE: Auto-printing of results may depend on the python interpreter you are using.

In [9]:

```
# Semicolon at the end of a statement is optional.
# If you don't put a semicolon and don't assign the result into a variable, th
# Jupyter shows the output only from the last statement (this behavior
#  depends on your particular Jupyter installation).
2+2
3+3
```

Out[9]:

```
6
```

```
In [10]: # Semicolon suppresses output.
         2+2;
```

```
In [5]: # Assigning to a variable also suppresses output (even without a semicolon)
        a=2+2
```

```
In [6]: # If you want to see the result, just have an additional statement with that v
        a=2+2
        a

Out[6]: 4
```

```
In [7]: # Ipython prints the result of only the last statement in a multi-line cell.
        # This behavior may depend on your version of Python/Ipython
        2+2
        3+3

Out[7]: 6
```

```
In [16]: # You can assign multiple variables at once.
         a=b=2+2
         a,b = 5,6
         a,b

Out[16]: (5, 6)
```

```
In [17]: a=5;
         # Use print() to display values
         print(a)
         # Use + to combine strings. But any non-string variable needs to be explicitly
         # converted using str(). See String Formatting below for other options.
         print('a is '+ str(a))

         5
         a is 5

Out[17]: '5.38780909211'
```

Indentation

Python is an "indented" language. Whereas you enclose block statements with the "end" keyword in Matlab; in Python you indent a block. When you need to end a block, just indent back. Remember to use a colon to start a block.

In [18]:

```
x=8; y=5;
if x%2==0:
    print('(A) x is even.')
    if y%2==0:
        print('(B) y is even.')
    else:
        print('(C) y is odd.')
else:
    print('(D) x is odd.')
    if y%2==0: print('(E) y is even.')
    else:
        print('(F) y is odd.')
```

```
(A) x is even.
(C) y is odd.
```

Exercise

Let x contain a grade between 0-100. print the letter grade corresponding to x, following the grading scale available here (http://instructorlink.berkeley.edu/centers/grading/chart.html). You may only do this for A+, A, A-, B+,B,B-, and consider everything else an F.

In [23]:

```
x=87;
```

Strings

In [24]:

```
# You can create strings with single quotes or double-quotes.
s='do not';
s="do not";
# You need to escape any quote characters with a backslash
s='don\'t';
# But you don't have to escape a single quote within a double quote (and vice
s="don't";
s='"ahmet" sacan';

# The usual escape sequences work, e.g., \n for a newline and \t for tab.
# In matlab, these escape sequences are not natively available, you had to use
s="first line\nsecond line";

# Break-up a long string string by a backslash (make sure there are no spaces
s="first line\n\
second line";
# Three double quotes can be used to write a long multi-line string.
s="""first line
second line""";

# To create a "raw" string where backslashes are just backslashes, use r in fr
s=r"first line\nsecond line. this is literally \\\ three backslashes.";
s
```

Out[24]:

```
'first line\\nsecond line. this is literally \\\\\\ three backslashes.'
```

In [12]:

```
# To get the length of a string, use len(). this is similar to Matlab's numel(
len(s)
```

Out[12]:

```
23
```

In [13]:

```
# You can concatenate strings with a plus sign or space. Don't enclose them in
s='hello ' + 'world'
s='hello ' 'world'
s
```

Out[13]:

```
'hello world'
```

In [14]:

```
# Replicate a string with *
s*5
```

Out[14]:

```
'hello worldhello worldhello worldhello worldhello world'
```

Indexing Strings

In [1]:

```
s="apple_orange_banana";

# Python uses 0-based indexing. To access the first element, use 0.
s[0]
```

Out[1]:

```
'a'
```

In [16]:
```
# To select a range of elements, use the colon operator. The upper index is no
s[0:3]
```

Out[16]:
```
'app'
```

In [17]:
```
# Leave out the lower bound to mean zero. Leave out the upper bound to mean th
s[:2]  %just a shorthand for s[0:2]
```

Out[17]:
```
'ap'
```

In [18]:
```
s[2:] %just a shorthand for s[2:numel(s)]
```

Out[18]:
```
'ple_orange_banana'
```

In [19]:
```
# You can use a negative number to index from the end of string
s[-1] # last character
```

Out[19]:
```
'a'
```

In [20]:
```
s[-2:] # last two characters. same as s[-2:-1]
```

Out[20]:
```
'na'
```

In [21]:
```
s[:-2] # everything but last two characters
```

Out[21]:
```
'apple_orange_bana'
```

In [26]:
```
# Strings are read-only. You can not change an individual character of a strin
#s[0]='x'; #ERROR
```

Exercise

Replace the first element of s with 'x'.

In [31]:
```

```

Out[31]:
```
'xpple_orange_banana'
```

String Functions

In [2]:
```
print(s)
# Strings are objects. The functions are available with a dot.
s.find('or') # search for a string. returns the index
```

```
apple_orange_banana
```

Out[2]:
```
6
```

In [24]:
```
# Strings are objects in python, and they come with their own pre-defined meth
s.startswith('app')
```

Out[24]:
```
True
```

In [25]:
```
s.endswith('zzz')
```

Out[25]:
```
False
```

In [26]:
```
s.replace('orange','raisin') #this doesn't replace in s, but returns a new str
```

Out[26]:
```
'apple_raisin_banana'
```

In [27]:
```
# To replace s itself, make an assignment
s = s.replace('orange','raisin')
s
```

Out[27]:
```
'apple_raisin_banana'
```

In [28]:
```
# Other String functions:
'abc123'.isalnum() #check whether string is composed of all alphanumeric chara
```

Out[28]:
```
True
```

In [29]:
```
'123'.isdigit()    #check whether string is composed of all digits.
```

Out[29]:
```
True
```

In [30]:
```
'abc'.isspace()    #allspaces?
```

Out[30]:
```
False
```

In [31]:
```
'abc'.isupper()    #all uppercase?
```

Out[31]:
```
False
```

In [32]:
```
'abc'.islower()    #all lower?
```

Out[32]:
```
True
```

In [33]:
```
#convert to lower case (does not change the string itself).
# matlab: upper('ApPlE')
'ApPlE'.lower()
```

Out[33]:
```
'apple'
```

In [34]:
```
# convert to upper case (does not change the string itself).
# matlab: upper('ApPlE')
'ApPlE'.upper()
```

Out[34]:
```
'APPLE'
```

```
In [35]: # remove any whitespace from the beginning and end of string.
         # matlab: strtrim('  apple   ')
         '  apple   '.strip()
```

```
Out[35]: 'apple'
```

```
In [36]: # split string into words (use additional argument if you want to split by oth
         # matlab: strsplit('apple orange banana')
         'apple orange banana'.split()
```

```
Out[36]: ['apple', 'orange', 'banana']
```

```
In [37]: 'app le,5,7'.split(',')
```

```
Out[37]: ['app le', '5', '7']
```

```
In [38]: # split string into lines.
         # matlab: strsplit(sprintf('apple\norange\nbanana'),sprintf('\n'))
         'apple\norange\nbanana'.splitlines()
```

```
Out[38]: ['apple', 'orange', 'banana']
```

```
In [39]: # join a set of strings with a character.
         # matlab: strjoin({'apple', 'orange', 'banana'}, '|')
         '|'.join(['apple', 'orange', 'banana'])
```

```
Out[39]: 'apple|orange|banana'
```

String Formatting

```
 In [3]: # Use % similar to sprintf. Remember to enclose the items in ().
         # matlab: sprintf('%f celcius = %f fahrenheit', 20, -6.67)
         '%.3f celcius = %.5f fahrenheit' % (20,-6.67)
```

```
 Out[3]: '20.000 celcius = -6.67000 fahrenheit'
```

```
In [41]: # Or use '{}'.format(), which acts similar to %.
         '{0}, {1}, {2}, {0}'.format('a', 'b', 'c')
```

```
Out[41]: 'a, b, c, a'
```

```
In [42]: '{0:05.2f}/{1:03d}/{2:04d}'.format(0.3333,2,3)
```

```
Out[42]: '00.33/002/0003'
```

In [9]:

```
print('========== Without Widths:')
x=1; print( '{0:d} {1:d} {2:d} {2:d}'.format(x, x*x, x*x*x, x*x*x*x*x*x) )
x=2; print( '{0:d} {1:d} {2:d} {2:d}'.format(x, x*x, x*x*x, x*x*x*x*x*x) )
x=3; print( '{0:d} {1:d} {2:d} {2:d}'.format(x, x*x, x*x*x, x*x*x*x*x*x) )
x=4; print( '{0:d} {1:d} {2:d} {2:d}'.format(x, x*x, x*x*x, x*x*x*x*x*x) )

print('\n========== With Widths:')
# The width is useful when printing a table, so numbers line up.
x=1; print( '{0:2d} {1:3d} {2:5d} {3:7d}'.format(x, x*x, x*x*x, x*x*x*x*x*x) )
x=2; print( '{0:2d} {1:3d} {2:5d} {3:7d}'.format(x, x*x, x*x*x, x*x*x*x*x*x) )
x=3; print( '{0:2d} {1:3d} {2:5d} {3:7d}'.format(x, x*x, x*x*x, x*x*x*x*x*x) )
x=4; print( '{0:2d} {1:3d} {2:5d} {3:7d}'.format(x, x*x, x*x*x, x*x*x*x*x*x) )
```

```
========== Without Widths:
1 1 1 1
2 4 8 8
3 9 27 27
4 16 64 64

========== With Widths:
 1   1     1       1
 2   4     8      64
 3   9    27     729
 4  16    64    4096
```

String to Number Conversion

In [44]:

```
# use float() to convert a string to a number.
# matlab: str2double('3.5')
float('3.5')
```

Out[44]:

```
3.5
```

In [45]:

```
# Whereas Matlab's str2double() returns NaN for things it cannot parse, float(
#float('A3.5')  #ERROR!
```

In [46]:

```
# If you know the string has an integer, you can convert it to int:
# matlab: str2double('4')
int('4')
```

Out[46]:

```
4
```

In [47]:

```
# To convert a number to a string, use the string formatting described above,
# matlab: sprintf('price: $%f',3.57)
# matlab: [ 'price: $' num2str(3.57) ]
'price: $' + str(3.57)
```

Out[47]:

```
'price: $3.57'
```

Exercise

Take a,b,c which are string variables containing a single digit each. Find the numerical value of abc.

In [48]:

```
#define some example values for a,b,c
a='5'; b='9'; c='3';

#write an expression that uses a,b,c and stores 593 as a number into variable
```

Exercise: wrap with html tag

Write code that starts with a string s and an html element name elem, and evaulates s.

In [13]:

```
#e.g.:
s='apple';
elem='b'; #the "bold" text tag in html.

#write an expression that returns <b>apple</b>
```

Out[13]:

```
'<b>apple</b>'
```

Error Handling

You often don't want to stop the entire program for small errors you can do something about. Python allows to handle errors similar to Matlab's try...catch....

In [17]:

```
s='A 4.5'
f=float('nan');
try:
    f=float(s)
    print('Converted string "'+s+'" to float ['+str(f)+']')
except Exception as e:
    print('--- ERROR: Exception encountered: '+str(e))

print('I am here')
#matlab: try ....; catch me; .... end
f
```

```
--- ERROR: Exception encountered: could not convert string to float: 'A 4.5'
I am here
```

Out[17]:

```
nan
```

Tuples, Lists, Dictionaries

- **Tuples and Lists** can be considered as **Matlab cell-arrays**.
- Tupples are "immutable" (You cannot change its elements once you create them), Lists are not.
- **Dictionaries** are like **Matlab structs**.

In [51]:

```python
# Use paranthesis to create a "tuple" (immutable).
a=('Jan','Feb','Mar','Apr');

# Use [] to index a tuple. Remember 0-based indexing!
a[1:2]
```

Out[51]:

```
('Feb',)
```

In [52]:

```python
# You cannot change elements of a tuple!
# a[0]='January' #ERROR!
# If you do want to change an element, construct a new tuple:
a = ('January',a[2],a[3])
a
```

Out[52]:

```
('January', 'Mar', 'Apr')
```

In [53]:

```python
# Create a List with [].
a=['apple','orange','grape'];
a[2]='raisin' #You can change an element of a List.
a
```

Out[53]:

```
['apple', 'orange', 'raisin']
```

In [54]:

```python
# Unlike Matlab, the List is not extended upon assignment.
# a[3]='banana' #ERROR! the indexed entry does not exist.

# You should use append() function to add a new entry.
# Note that append() changes the List itself. No need to assign it with a=...
a.append('banana')
a
```

Out[54]:

```
['apple', 'orange', 'raisin', 'banana']
```

In [55]:

```python
# You can also use '+' operator to append to a list or to combine two lists.
b = a + ['cherry','berry']
b
```

Out[55]:

```
['apple', 'orange', 'raisin', 'banana', 'cherry', 'berry']
```

In [56]:

```python
# Create a dictionary with {field1: value, field2:value2, etc.}
# Note that you don't have the field naming constraints (alphanumeric) Matlab
# You can use strings or numbers as fields.
a={'apple':'red', 5:10, 'orange banana':100}
a
```

Out[56]:

```
{'apple': 'red', 5: 10, 'orange banana': 100}
```

In [57]:

```python
# To access a dictionary field's value, use [fieldname]
a['apple'], a[5]
```

Out[57]:

```
('red', 10)
```

Exercise

Write code that takes a url variable and returns the hostname part of the url.

In [19]:
```
url='https://docs.python.org/2/library/string.html';
#write code to extract 'docs.python.org' from url
```

Out[19]:
```
'docs.python.org'
```

List Functions

In [24]:
```
# Remember to use comma to separate items in a list, otherwise you would be co
a=['apple' 'orange' 'banana' 'blueberry' 'blackberry' 'strawberry']
a
```

Out[24]:
```
['appleorangebananablueberryblackberrystrawberry']
```

In [25]:
```
a=['apple', 'orange', 'banana','blueberry','blackberry','strawberry']
a.append('raisin') #add en element to the end of the list
a
```

Out[25]:
```
['apple',
 'orange',
 'banana',
 'blueberry',
 'blackberry',
 'strawberry',
 'raisin']
```

In [26]:
```
a.insert(2,'grape') #insert at a specific position
a
```

Out[26]:
```
['apple',
 'orange',
 'grape',
 'banana',
 'blueberry',
 'blackberry',
 'strawberry',
 'raisin']
```

In [27]:
```
a.remove('apple') #find and remove a  specific item. Will throw ERROR! if item
a
```

Out[27]:
```
['orange',
 'grape',
 'banana',
 'blueberry',
 'blackberry',
 'strawberry',
 'raisin']
```

```
In [28]: a.pop()  #remove the last element
         a
```

```
Out[28]: ['orange', 'grape', 'banana', 'blueberry', 'blackberry', 'strawberry']
```

```
In [29]: a.pop(1) #remove the element at a given position
         a
```

```
Out[29]: ['orange', 'banana', 'blueberry', 'blackberry', 'strawberry']
```

```
In [30]: a.index('blackberry') #find and return the index of an item. ERROR! if item is
```

```
Out[30]: 3
```

```
In [33]: # sort the list. it'll use alphabetical order for strings, and numerical order
         # ERROR! if it doesn't know how to compare items (e.g., if you mix numbers and
         a.sort()
         a
```

```
Out[33]: ['banana', 'blackberry', 'blueberry', 'orange', 'strawberry']
```

```
In [34]: a.reverse() #reverse the order of the elements
         a
```

Defining Functions

```
In [35]: # Use "def" keyword to define a function.
         # You need to list the inputs to a function, but Unlike Matlab, the output var
         # Unlike matlab, you can only return a single thing (which can be a List of ma
         def isodd(x):
             if x%2==0:
                 return False
             else:
                 return True


         #Function definition "ends" when you move back in indentation.
         isodd(17), isodd(22), isodd(33)
```

```
Out[35]: (True, False, True)
```

In [39]:

```
# Describe what your function does using """..."""
def isodd(x):
    """Return True if x is odd, False otherwise"""
    return x%2==1

# Note that the documentation of any function is available using the __doc__ p
# Yes, functions are objects, like almost anything else in Python.
isodd.__doc__
```

Out[39]:

```
'Return True if x is odd, False otherwise'
```

In [70]:

```
# If function contains a single statement, define it on a single line.
def isodd(x): return x%2==1

isodd(17), isodd(22), isodd(33)
```

Out[70]:

```
(True, False, True)
```

In [71]:

```
# You can specify default arguments for function arguments. No more "if ~exist
def fun(a=10, b=20): return a*b

fun(), fun(2), fun(2,5)
```

Out[71]:

```
(200, 40, 10)
```

In [72]:

```
# When calling a function, you can use the named arguments in any order.
def fun(a, b): return a*b

fun(5,2), fun(b=3,a=4), fun(10,b=3)
```

Out[72]:

```
(10, 12, 30)
```

In [73]:

```
# WARNING: Default arguments are assigned once if they are objects (they endup
def fun(a, q=[]):
    q.append(a)
    return q

print( fun(10) )
print( fun(20) )
print( fun(30) )
```

```
[10]
[10, 20]
[10, 20, 30]
```

In [74]:

```
# If you don't like that behavior, use "None" as default value, so that variab
# None every time you call the function.
def fun(a, q=None):
    if q is None: q=[]
    q.append(a)
    return q

print( fun(10) )
print( fun(20) )
print( fun(30) )
```

```
[10]
[20]
[30]
```

In [41]:

```
# Use * and ** to capture additional unnamed (similar to varargin in Matlab) a
def fun(a,b, *unnamed, **named):
    print('a=' + str(a) + ', b=' + str(b))
    print('unnamed arguments: ' + str(unnamed))
    print('named arguments: ' + str(named))

fun(3,4, 5,6, x=10, y=20, z=30)
```

```
a=3, b=4
unnamed arguments: (5, 6)
named arguments: {'x': 10, 'y': 20, 'z': 30}
```

In [76]:

```
# You can "unpack" a list to be passed as arguments to a function using *. (si
def fun(a,b): return a*b

x=[3,5]
fun(*x)
```

Out[76]:

```
15
```

Exercise: normpdf()

Write a function normpdf(x, avg, std) that returns the Guassian probabily density function value of x for a normal distribution with mean avg and standard deviation std. If avg is not given, use 0. If std is not given, use 1. See Normal Distribution @wikipedia (https://en.wikipedia.org/wiki/Normal_distribution) for the formula of Gaussian probability density function

In [2]:

```



print(normpdf(5,3,1.5))
print(normpdf(5,3))
print(normpdf(5))
print(normpdf(std=0.5,x=4))
```

```
0.10934004978399577
0.05399096651318806
1.4867195147342977e-06
1.0104542167073785e-14
```

List Comprehension

In [77]:

```
# You can go through a list without having to write a for loop.
a=[5,10,15,20,25,30];

# Go through each element of a, apply x*x, and collect results in a new List.
b = [x*x for x in a]
b
```

Out[77]:

```
[25, 100, 225, 400, 625, 900]
```

In [78]:

```
# What you do can be any complex expression. The result will be a list of what
b = [[x, 2*x, x*x] for x in a]
b
```

Out[78]:

```
[[5, 10, 25],
 [10, 20, 100],
 [15, 30, 225],
 [20, 40, 400],
 [25, 50, 625],
 [30, 60, 900]]
```

In [79]:

```
# Go through elements that match a certain criteria (odd elements in this exam
b = [x*x for x in a  if x%2==0]
b
```

Out[79]:

```
[100, 400, 900]
```

In [3]:

```
# Use the range() function when you need a list of consequtive numbers
#Create a List of numbers between 0 (inclusive) and 100 (exclusive) with step
[x for x in range(0,100,20)]
```

Out[3]:

```
[0, 20, 40, 60, 80]
```

In [4]:

```
# python delays the evaluation of range() until it is needed.
a=range(0,100,20)
a
```

Out[4]:

```
range(0, 100, 20)
```

Exercise: word lengths

Given a list of words, find the length of each word using list comprehension.

```
In [5]:  words = ['peter','piper','picked','a','peck','of','pickled','peppers'];

         #find the lengths of each word, and store in a list called wordlengths.

Out[5]:  [5, 5, 6, 1, 4, 2, 7, 7]
```

Exercise: filter positive

Given a list of numbers, create a new list that contains only the positive numbers selected from the original list. Use list comprehension.

```
In [9]:  numbers = [-1, -10, 3, 5, 9, -7, 5, 8, -4]

Out[9]:  [3, 5, 9, 5, 8]
```

for loop

```
In [83]: # Use the "in" keyword (instead of the "=" in Matlab)
         for x in range(0,5):
             print(x)

         0
         1
         2
         3
         4
```

```
In [10]: # A huge advantage over Matlab is that you can iterate over anything, not just
         for x in ['apple',120,'orange','banana']:
             print(x)

         apple
         120
         orange
         banana
```

```
In [13]: # To simultaneously iterate over keys and values of a dictionary:
         a={'x':5, 'y':10};
         for key,val in a.items():
             print('key='+key+', val='+str(val))

         key=x, val=5
         key=y, val=10
```

In [86]:

```
# Equivalently, iterate over the keys and get the value inside the for loop.
a={'x':5, 'y':10};
for key in a:
    val=a[key]
    print('key='+key+', val='+str(val))
```

```
key=x, val=5
key=y, val=10
```

Exercise: word lengths

Given a list of words, find the length of each word using for loop.

In [87]:

```
words = ['peter','piper','picked','a','peck','of','pickled','peppers'];

#find the lengths of each word, and store in a list called wordlengths. Use fo
```

Python packages

In [15]:

```
# Beyond the basic Python functions, you need to know which package a function
# that package before you can use its functions. (What a bother!)

#import the socket package. its functions will be accessible as socket.functio
import socket

print(socket.gethostname())
```

```
sacanlap2
```

In [89]:

```
# if typing "socket" is more than you are willing to type, give it a short nam
import socket as sock

print(sock.gethostname())
```

```
sacanlap2
```

In [90]:

```
# if you don't want to "dot" it everytime, import all the functions to be avai
from socket import *
# now gethostname() and all the other functions from the socket package become

print(gethostname())
```

```
sacanlap2
```

```
In [91]: # The packages you try to import need to be already installed on your computer
         # import ahmet_nonexistentpackage #ERROR!: No module named 'ahmet_nonexistentp

         # When you need to install a new package, consult google for how to install it
         # It usually involves running a command in the terminal/commandwindow.
```

Exercise

Get the path of the temporary directory.

```
In [ ]:
```

```
In [ ]:
```

Numpy

```
In [18]: # Numerical computing functionality is available from the numpy package.
         # numpy doesn't come builtin to a python installation (but Ipython comes prepa

         import numpy as np

         # np.array() is how you can create a numerical vector.
         x = np.array( [6, 8, 77, 55, 23, 17, 19] )
         x

Out[18]: array([ 6,  8, 77, 55, 23, 17, 19])
```

```
In [19]: # The usual python indexing rules apply here.
         # Remember that indexes start from 0.
         # A negative index i means end+i
         x[0] #x[ind]

Out[19]: 6
```

```
In [94]: x[0:4] #x[from:uptonotincluding]

Out[94]: array([ 6,  8, 77, 55])
```

```
In [95]: x[0:4:2] #x[from:uptonotincluding:step]

Out[95]: array([ 6, 77])
```

```
In [96]: x[3:] #x[from:] to the end.

Out[96]: array([55, 23, 17, 19])
```

In [97]:

```
x[:3] #x[:to] from the beginning.
```

Out[97]:

```
array([ 6,  8, 77])
```

In [98]:

```
x[:-1]
```

Out[98]:

```
array([ 6,  8, 77, 55, 23, 17])
```

Exercise

Create a 1x10 random vector and find the index and the value of the smallest number.

In [22]:

```
#create a random vector
import numpy as np
```

Out[22]:

```
array([[0.60592026, 0.74237815, 0.02246163, 0.28721324, 0.04530202,
        0.54224833, 0.01404789, 0.43993012, 0.5340141 , 0.14224916]])
```

In [28]:

```
#find the index & value of smallest number
```

```
[6, 0.014047885419249728]
```

Matrix Indexing

In [32]:

```
# To create a matrix, use a separate list for each row.
x = np.array([[1, 2], [3, 4], [5, 6]])
x
```

Out[32]:

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [33]:

```
# Unlike Matlab, the following row-column indexing is done in pairs.
# i.e. it's selecting elements at positions: <0,0>,<1,1>,<2,0>
x[[0, 1, 2], [0, 1, 0]]
```

Out[33]:

```
array([1, 4, 5])
```

In [34]:

```
# Colon operator behaves the same, it means select all rows (or columns)
x[:,1]
```

Out[34]:

```
array([2, 4, 6])
```

In [35]:

```
x[1,:]
```

Out[35]:

```
array([3, 4])
```

In [105]:

```
# WARNING: Indexing creates a "view" of the original matrix.
x = np.array([[1, 2, 3], [4, 5, 6]])
y=x[0,:]  #y is "attached" to elements of x.
print('Before changing y:\n x='+str(x)+'\n, y='+str(y))
y[0]=99   #this also changes x.
print('\nAfter changing y:\n x='+str(x)+'\n, y='+str(y))
```

```
Before changing y:
 x=[[1 2 3]
 [4 5 6]]
, y=[1 2 3]

After changing y:
 x=[[99  2  3]
 [ 4  5  6]]
, y=[99  2  3]
```

In [106]:

```
# If you want to create a copy, use the copy() function.
x = np.array([[1, 2, 3], [4, 5, 6]])
y=x[0,:].copy()
print('Before changing y:\n x='+str(x)+'\n, y='+str(y))
y[0]=99   #this also changes x.
print('\nAfter changing y:\n x='+str(x)+'\n, y='+str(y))
```

```
Before changing y:
 x=[[1 2 3]
 [4 5 6]]
, y=[1 2 3]

After changing y:
 x=[[1 2 3]
 [4 5 6]]
, y=[99  2  3]
```

Matrix Size & Shape

In [36]:

```
x = np.array([[1, 2], [3, 4], [5, 6]])
x.shape  #Get the number of rows/columns of matrix. --similar to Matlab's size
```

Out[36]:

```
(3, 2)
```

In [38]:

```
x.size  #Get number of elements. --similar to Matlab's numel().
```

Out[38]:

```
6
```

In [109]:

```
y = x.reshape(1,6)  #--similar to Matlab's reshape(). Does not change x itself
y
```

Out[109]:

```
array([[1, 2, 3, 4, 5, 6]])
```

In [110]:

```
print('Before resize:\n'+str(x))
x.resize(2,3)  #same as reshape, but reshapes in-place.
print('\nAfter resize:\n'+str(x))
```

```
Before resize:
[[1 2]
 [3 4]
 [5 6]]

After resize:
[[1 2 3]
 [4 5 6]]
```

In [111]:

```
np.tile(x,[2,3]) #--similar to Matlab's repmat().
```

Out[111]:

```
array([[1, 2, 3, 1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6, 4, 5, 6],
       [1, 2, 3, 1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6, 4, 5, 6]])
```

In [112]:

```
x.repeat(3,axis=0)  #--repeat elements along a dimension.
```

Out[112]:

```
array([[1, 2, 3],
       [1, 2, 3],
       [1, 2, 3],
       [4, 5, 6],
       [4, 5, 6],
       [4, 5, 6]])
```

In [113]:

```
x.repeat([3],axis=1)  #--repeat elements along a dimension.
```

Out[113]:

```
array([[1, 1, 1, 2, 2, 2, 3, 3, 3],
       [4, 4, 4, 5, 5, 5, 6, 6, 6]])
```

Exercise

Create an 8x8 checkerboard of values 0 & 1.

In [ ]:

```

```

Matrix Operations

In [114]:

```
# Python lists do not natively have matrix operations
[1,2,3] * 3
```

Out[114]:

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

In [115]:
```
# Unlike Matlab, multiplication, division, and power operators work element-wi
np.array([2,4,8]) * np.array([10,100,1000])
```

Out[115]:
```
array([  20,  400, 8000])
```

In [140]:
```
#if you need matrix multiplication, use dot()
np.dot(x,np.transpose(x))
```

Out[140]:
```
1688.2218054494956
```

Statistics

The scipy.stats package contains many of the statistics functions you would need.

In [147]:
```
from scipy import stats

#two tailed ttest for testing whether two independent sample sets are statisti
(_,pvalue)=stats.ttest_ind([1,2,3,4,5],[11,12,13,14])
pvalue
```

Out[147]:
```
2.6601397199213477e-05
```

In [150]:
```
#non-parametric test
(_,pvalue)=stats.ranksums([1,2,3,4,5],[11,12,13,14])
pvalue
```

Out[150]:
```
0.014305878435429648
```

In [152]:
```
#non-parametric paired test, e.g., comparing mesurements before & after treatm
(_,pvalue)=stats.wilcoxon([1,2,3,4,5],[11,12,13,14,15])
pvalue
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\morestats.py:2397: Use
rWarning: Warning: sample size too small for normal approximation.
  warnings.warn("Warning: sample size too small for normal approximation.")
```

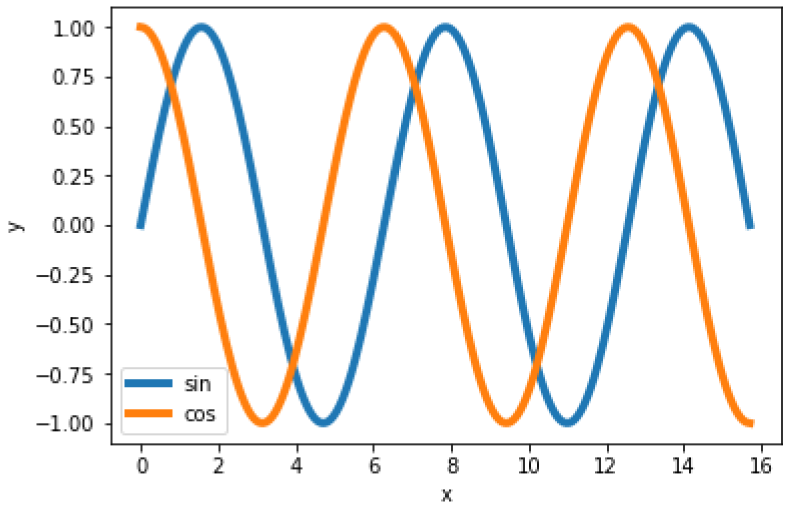
Out[152]:
```
0.025347318677468252
```

Plotting

In [1]:
```
# Plotting functionality is available from matplotlib package.
# pylab is another package that exposes matplotlib and numpy functions.
# Import the pylab package to make the Matlab-like functions directly availabl
from pylab import *

# let's have numpy also available..
import numpy as np

# Also tell Jupyter to show plots within the notebook
%matplotlib inline
```

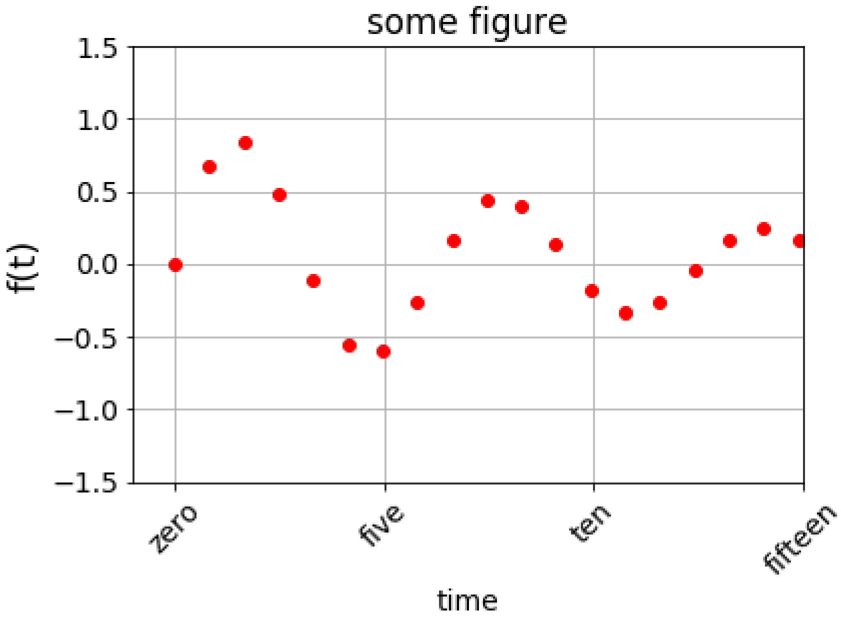
In [2]:

```
x = linspace(0,5*pi, 200);
plot(x, sin(x), x,cos(x), linewidth=4.0);
xlabel('x'); ylabel('y'); legend(['sin','cos']);
```



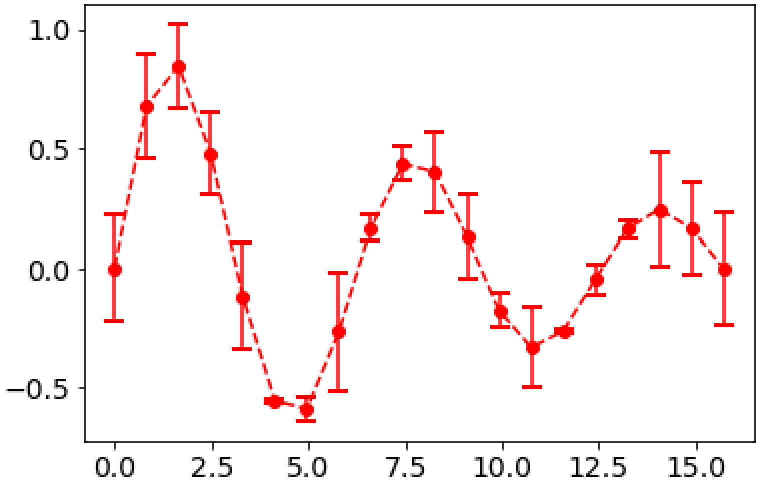
In [4]:

```
# To globally change font size. This affects any future plot you (re)generate
rcParams['font.size']=14;
rcParams['xtick.labelsize']=14;
rcParams['ytick.labelsize']=14;


x = linspace(0,5*pi, 20);
#change the line style to get a scatter plot
plot(x, exp(-x/10)*sin(x), 'ro');

# Set labels and titles. fontsize can be individually specified.
title('some figure');
xlabel('time');
ylabel('f(t)', fontsize=18);

#change the axis limits:
xlim(-1,15);
ylim(-1.5,1.5);

#change the x ticks:
xticks([0,5,10,15],['zero','five','ten','fifteen'],rotation=45);

# turn the grid on
grid();
```



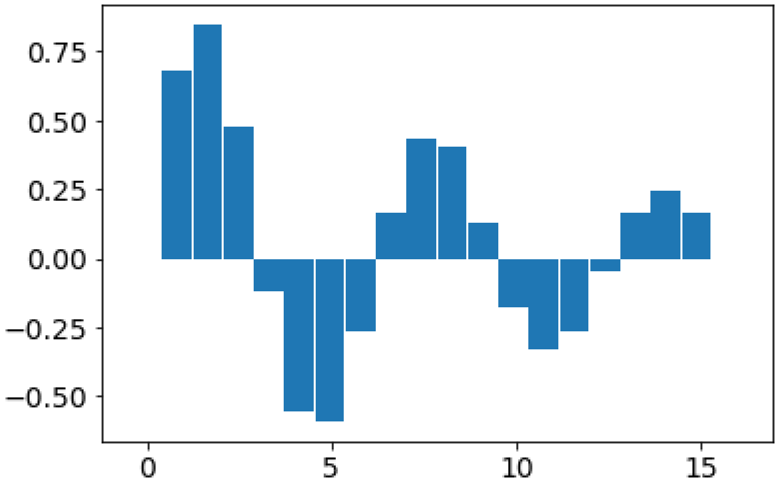
In [6]:

```
x = linspace(0,5*pi, 20);
y = exp(-x/10)*sin(x);
#change the line style to get a scatter plot
plot(x, y, 'r--o');

# errorbars:
errorbar(x, y, yerr=rand(x.size)/4, linestyle="none",color="red", capsize=5, m
```



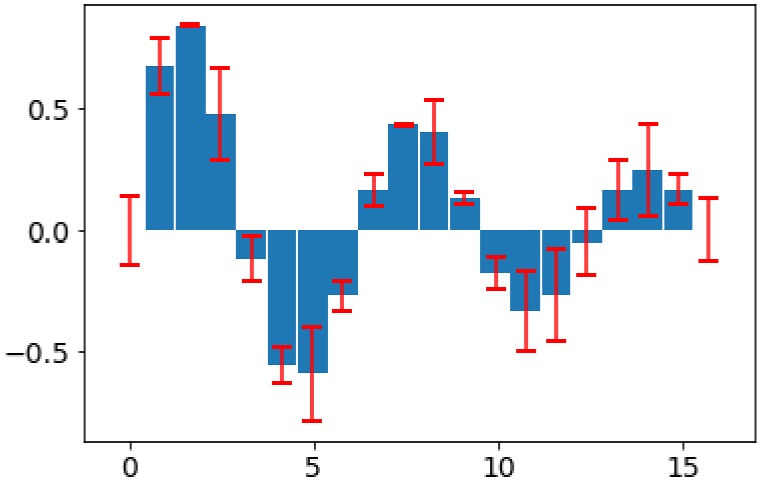
In [7]:

```
#bar plot
bar(x, y);
```



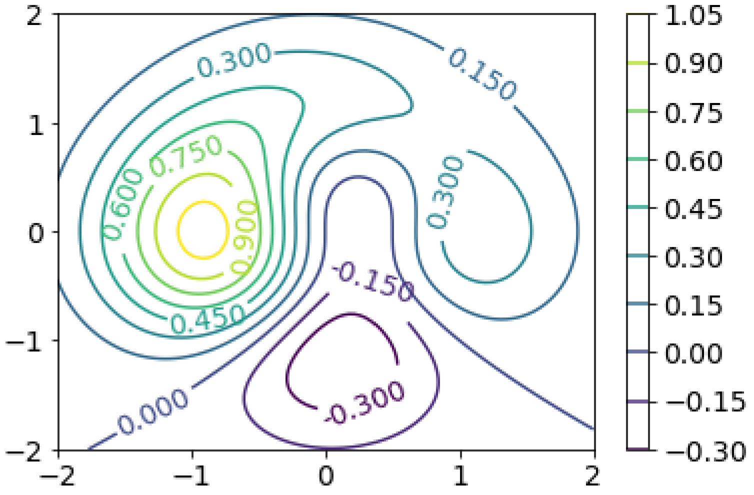
```
In [12]: #bar plot with error bars
         errorbar(x, y, yerr=rand(x.size)/4, linestyle="none",color="red", markeredgewi
         bar(x, y, zorder=5)
```

```
Out[12]: <Container object of 20 artists>
```



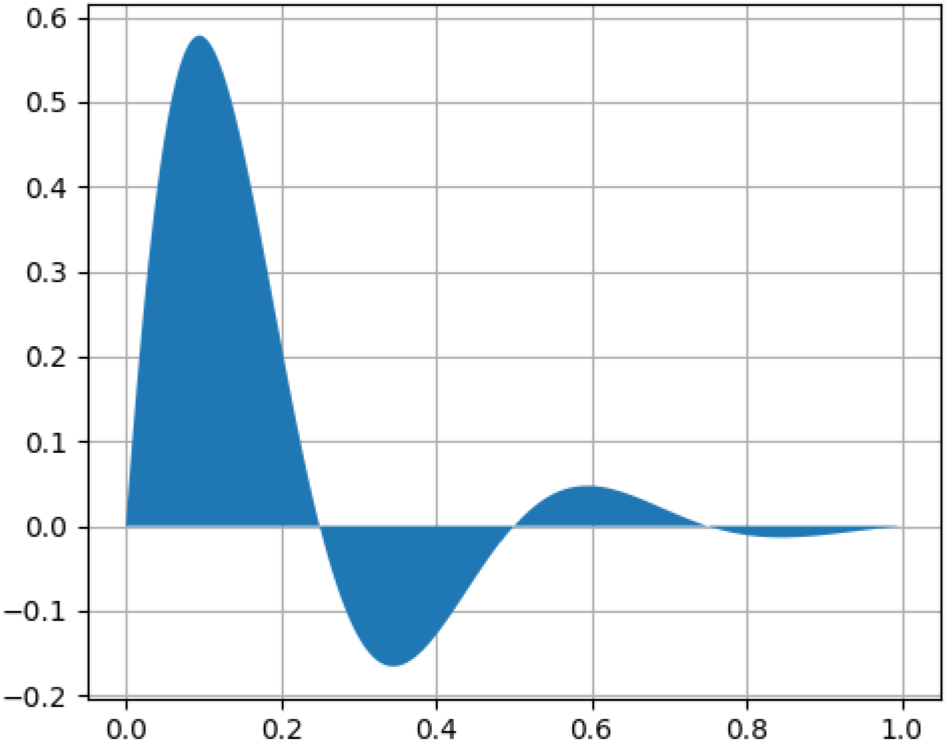
```
In [13]: #Countour plot
         def f(x,y): return (2*x**2 - x +y**3)*exp(-(x**2+y**2))
         x=y=linspace(-2,2,100)
         X,Y=meshgrid(x,y)
         h=contour(X,Y, f(X,Y), 10)
         clabel(h, inline=1)
         colorbar(h,orientation='vertical')
```

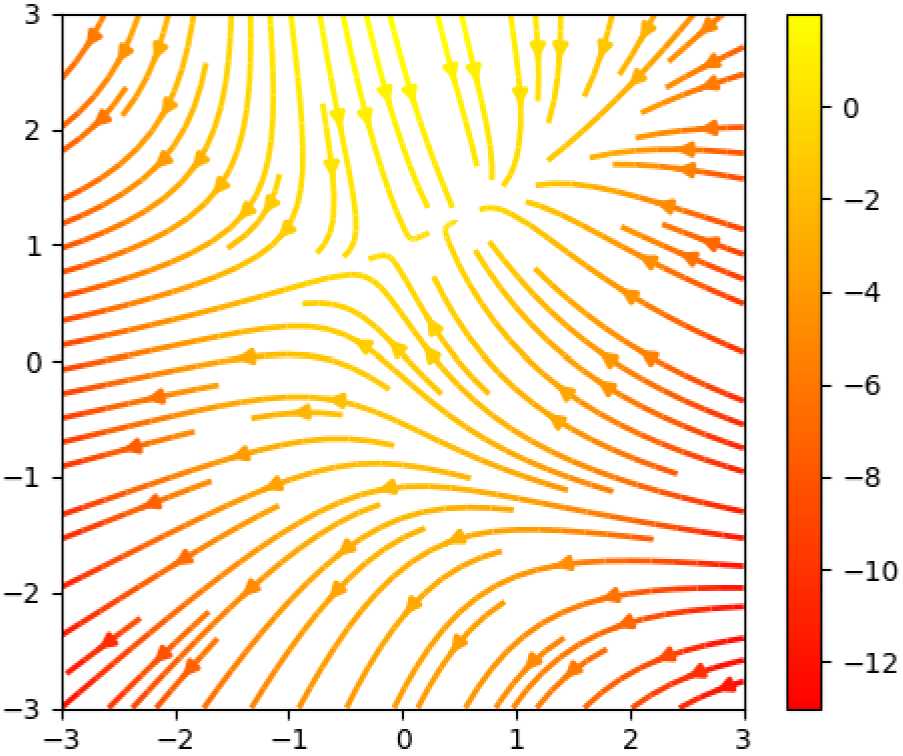
```
Out[13]: <matplotlib.colorbar.Colorbar at 0x1af52e6a9b0>
```

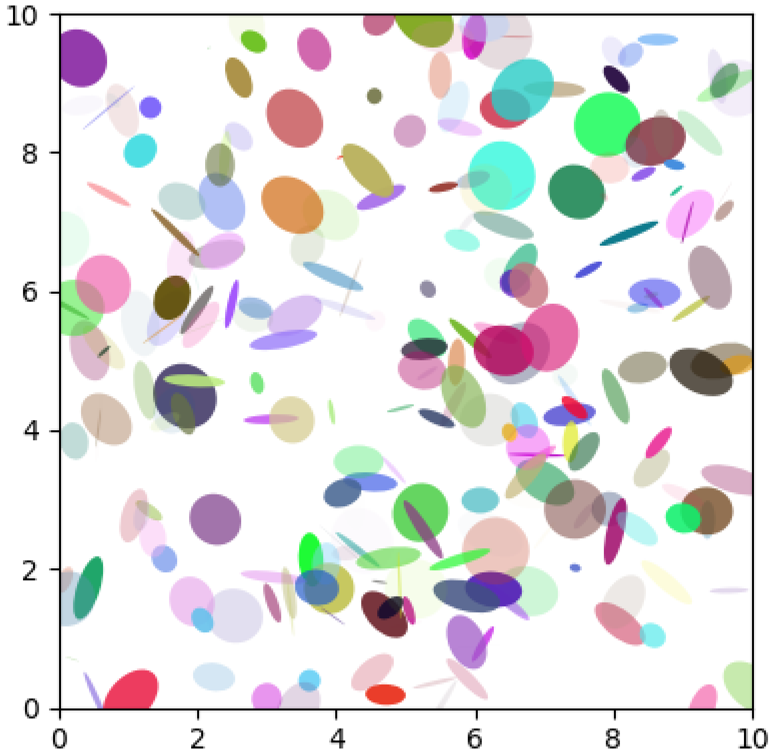


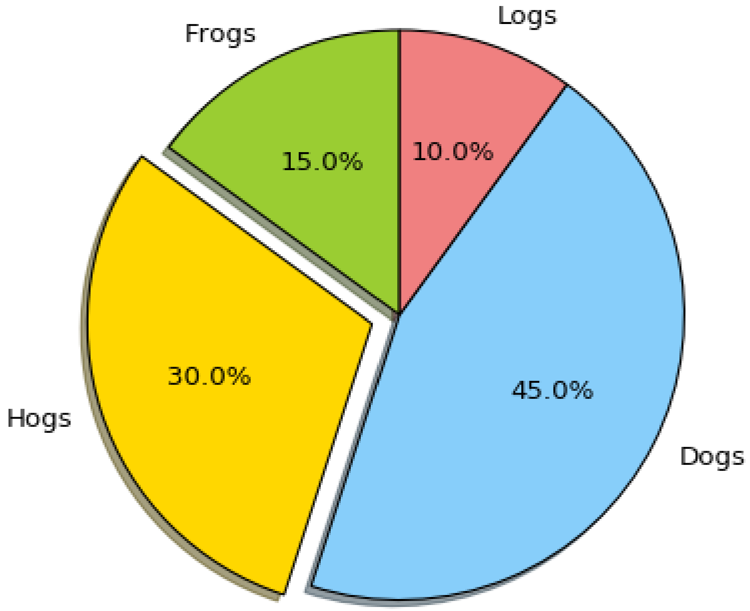
More examples

For more examples, see: http://matplotlib.org/users/screenshots.html (http://matplotlib.org/users/screenshots.html)






10
8
6
4
2
0
0
2
4
6
8
10


Frogs
Logs
15.0%
10.0%
30.0%
45.0%
Hogs
Dogs