In [1]:

```
%pylab inline
%config InlineBackend.figure_format = 'retina'
from ipywidgets import interact
```

```
Populating the interactive namespace from numpy and matplotlib
```

Python Objects

The purpose of this lesson is to give you a better idea of about the anatomy of Python objects. You may never need to create your own objects in this class, but if we remove a little of the mystery behind them, they will be easier to use.

Everything in python (function, variables, etc) is an object. Four our purposes, a Python object is a data structure, a 'container' that holds **atributes**, variables that hold data, and **methods**, functions that are attached to the object.

Atributes can be, for example, integers, floats, strings, arrays, lists etc.

The instructions for creating a certain type of object are called 'classes'. Instances of classes are called objects.

Below is a simple object with an atribute and a method.

In [46]:

```
class MyClass:
    def __init__(self): ## called once during construction of an instance
        ## `self` is the internal name of the object, which we can use to access any
        ## atributes or methods from within the object methods
        self.x = 2
    def compute_2_to_power_n(self, n):## the `self` argument is for internal use only
        return self.x**2
### you access atributes and methods with a '.' as in 'object_name.atribute_name'
C = MyClass() ## create object C, which is an instance of MyClass
print(C.x)
print(C.compute_2_to_power_n(2))
## notice I only used one input argument `n`
## the `self` argument is used only within the class itself
```

```
2
4
```

An array is an example of a Python object

In [2]:

```
x = array([1., 2.])
print(x)
print('These are atributes')
print(x.size, x.shape, x.ndim)
print('These are methods of the array object')
## notice that they are all functions
## even if they take no arguments, they still require '()'
print(x.sum(), x.prod(), x.max(), x.min())
```

```
[1. 2.]
Thse are atributes
2 (2,) 1
These are methods of the array object
3.0 2.0 2.0 1.0
```

In [3]:

```
A = array([[1., 2.], [0., -1.]])
print(A)
print('These are atributes')
print(A.size, A.shape, A.ndim)
print('there are many useful atributes, like the transpose 2d array')
print(A.T)
print('These are methods of the array object')
## notice that they are all functions
## even if they take no arguments, they still require '()'
print(x.sum(), x.prod(), x.max(), x.min())
print('diagonal of the 2d array')
print(A.diagonal())
```

```
[[ 1.  2.]
 [ 0. -1.]]
Thse are atributes
4 (2, 2) 2
there are many useful atributes, like the transpose 2d array
[[ 1.  0.]
 [ 2. -1.]]
These are methods of the array object
3.0 2.0 2.0 1.0
diagonal of the 2d array
[ 1. -1.]
```

Print the atributes and methods attached to an object

In [4]:

```
dir(x)
```

Out[4]:

```
['T',
 '__abs__',
 '__add__',
 '__and__',
 '__array__',
 '__array_finalize__',
 '__array_function__',
 '__array_interface__',
 '__array_prepare__',
 '__array_priority__',
 '__array_struct__',
 '__array_ufunc__',
 '__array_wrap__',
 '__bool__',
 '__class__',
 '__complex__',
 '__contains__',
 '__copy__',
```

You can also get information using the help system

In [5]:

```
help(x)
```

```
Help on ndarray object:

class ndarray(builtins.object)
 |  ndarray(shape, dtype=float, buffer=None, offset=0,
 |          strides=None, order=None)
 |
 |  An array object represents a multidimensional, homogeneous array
 |  of fixed-size items.  An associated data-type object describes the
 |  format of each element in the array (its byte-order, how many bytes it
 |  occupies in memory, whether it is an integer, a floating point number,
 |  or something else, etc.)
 |
 |  Arrays should be constructed using `array`, `zeros` or `empty` (refer
 |  to the See Also section below).  The parameters given here refer to
 |  a low-level method (`ndarray(...)`) for instantiating an array.
 |
 |  For more information, refer to the `numpy` module and examine the
 |  methods and attributes of an array.
 |
 |  Parameters
```

Want to know what a method does or how to use it? Use the help system

In [6]:

```
help(x.max)
```

```
Help on built-in function max:

max(...) method of numpy.ndarray instance
    a.max(axis=None, out=None, keepdims=False)

    Return the maximum along a given axis.

    Refer to `numpy.amax` for full documentation.

    See Also
    --------
    numpy.amax : equivalent function
```

In [7]:

```
help(numpy.amax)
```

```
Help on function amax in module numpy:

amax(a, axis=None, out=None, keepdims=<no value>, initial=<no value>)
    Return the maximum of an array or maximum along an axis.

    Parameters
    ----------
    a : array_like
        Input data.
    axis : None or int or tuple of ints, optional
        Axis or axes along which to operate.  By default, flattened input is
        used.

        .. versionadded:: 1.7.0

        If this is a tuple of ints, the maximum is selected over multiple axes,
        instead of a single axis or all the axes as before.
    out : ndarray, optional
        Alternative output array in which to place the result.  Must
```

Python dictionaries

A Python dictionary is another kind of container data structure. It uses (keyword, item) pairs. The 'keyword' is used to access the item so all the keywords need to be unique. The keywords can be many things, including strings, integers, and floats.

In [2]:

```
## d = {keyword1:item1, keyword2:item2, keyword3:item3, ...}
d = {'a': 1., 'b': 10} ## curly braces are for dictionaries
print(d)
## we access items similar to lists and arrays with square brackets '[]'
## but we use the keyword inside the brackets
print(d['a'])
print(d['b'])
```

```
{'a': 1.0, 'b': 10}
1.0
10

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-9f629c6d3ffd> in <module>
     15 b = 3.
     16
---> 17 some_function(b)
     18 a = arange(5)

<ipython-input-2-9f629c6d3ffd> in some_function(x)
     11     # I can only access x in this function
     12     y = 4. # exists only locally within this function
---> 13     return y*a
     14
     15 b = 3.

NameError: name 'a' is not defined
```

You can add elements to a dictionary after the dictionary has been constructed. Dictionary items can be any Python variable or object (including dictionaries)

In [7]:

```
## keywords 'c' and 'd' do not exist yet
## We can create them using simple item assignment
d['c'] = {'keyword1': arange(5)} ## a dictionary containing an array
d['d'] = rand(3) ## an array
print(d)

print('Print items one at a time')
for kw in d: ## loop over keywords to print items
    print(d[kw])
```

```
{'keyword1': array([0, 1, 2, 3, 4]), 'keyword2': -1, 'c': {'keyword1': array([0, 1,
2, 3, 4])}, 'd': array([0.87657442, 0.60629439, 0.55530273])}
Print items one at a time
[0 1 2 3 4]
-1
{'keyword1': array([0, 1, 2, 3, 4])}
[0.87657442 0.60629439 0.55530273]
```

Comparing dictionaries to lists

In [40]:

```
## A dictionary can mimic a list if you use integers as the keywrods
d = {0: 5., 1: 6., 2:0.}
l = [5., 6., 0.]
print(d)
print(l)
print(l[0], d[0])
print(l[1], d[1])
print(l[2], d[2])


## ... but it is important to remember that under the hood, lists and
## dictionaries are very different
print('iterating over list')
for lj in l:
    print(lj)
print('iterating over dictionary')
for dj in d: ## loops over the keywords, not the items
    print(dj)
```

```
{0: 5.0, 1: 6.0, 2: 0.0}
[5.0, 6.0, 0.0]
5.0 5.0
6.0 6.0
0.0 0.0
iterating over list
5.0
6.0
0.0
iterating over dictionary
0
1
2
```

In [41]:

```
## The keywords do not necessarily iterate in sorted order like a list
d = {1: 6., 2:0., 0: 5.}
for dj in d: ## loops over the keywords, not the items
    print(dj)
```

```
1
2
0
```

Assert Statements: good programming habits

Using `assert` statements will help you write better code. They will help you diagnose and fix bugs much faster. They also make your code more readable.

In [9]:

```
def my_function(x):
    assert x > 0 ## if this condition is not true, there will be an error
    return log(x)

def another_function(x, y):
    assert y != 0
    return x/y

print(my_function(12))
print(my_function(0.2))
print(another_function(1., 2.))
```

```
2.4849066497880004
-1.6094379124341003
0.5
```

In [10]:

```
my_function(-2)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-10-c3393f2e794a> in <module>
----> 1 my_function(-2)

<ipython-input-9-773127589589> in my_function(x)
      1 def my_function(x):
----> 2     assert x > 0 ## if this condition is not true, there will be an error
      3     return log(x)
      4
      5 def another_function(x, y):

AssertionError:
```

In [11]:

```
another_function(2, 0)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-11-045d4068b6bf> in <module>
----> 1 another_function(2, 0)

<ipython-input-9-773127589589> in another_function(x, y)
      4
      5 def another_function(x, y):
----> 6     assert y != 0
      7     return x/y
      8

AssertionError:
```

Chebpy

An external package for working with polynomial interpolation. https://github.com/chebpy/chebpy (https://github.com/chebpy/chebpy)

In [2]:

```
!pip install git+https://github.com/chebpy/chebpy.git
```

```
Collecting git+https://github.com/chebpy/chebpy.git
  Cloning https://github.com/chebpy/chebpy.git (https://github.com/chebpy/chebpy.gi
t) to /private/var/folders/g2/sbtthm856hg_gmhmcq5n09m80000gq/T/pip-req-build-3pcqcoe
c
  Running command git clone -q https://github.com/chebpy/chebpy.git (https://github.
com/chebpy/chebpy.git) /private/var/folders/g2/sbtthm856hg_gmhmcq5n09m80000gq/T/pip-
req-build-3pcqcoec
Requirement already satisfied: numpy>=1.16 in /anaconda3/lib/python3.7/site-packages
(from chebpy==0.4.1) (1.19.5)
Requirement already satisfied: pyfftw>=0.11 in /anaconda3/lib/python3.7/site-package
s (from chebpy==0.4.1) (0.12.0)
Building wheels for collected packages: chebpy
  Building wheel for chebpy (setup.py) ... done
  Created wheel for chebpy: filename=chebpy-0.4.1-py3-none-any.whl size=28619 sha256
=e128f43ed9d5c8c459350193751be4389e0604e22b156d5814a24e62b405e3f7
  Stored in directory: /private/var/folders/g2/sbtthm856hg_gmhmcq5n09m80000gq/T/pip-
ephem-wheel-cache-kqw57u5x/wheels/04/7c/df/d9e4d188fa2ed99773bc1ebe8d78f13f25ba2bd8c
683da3347
Successfully built chebpy
Installing collected packages: chebpy
  Attempting uninstall: chebpy
    Found existing installation: chebpy 0.2.0
    Uninstalling chebpy-0.2.0:
      Successfully uninstalled chebpy-0.2.0
Successfully installed chebpy-0.4.1
WARNING: You are using pip version 21.0.1; however, version 21.2.4 is available.
You should consider upgrading via the '/anaconda3/bin/python -m pip install --upgrad
e pip' command.
```

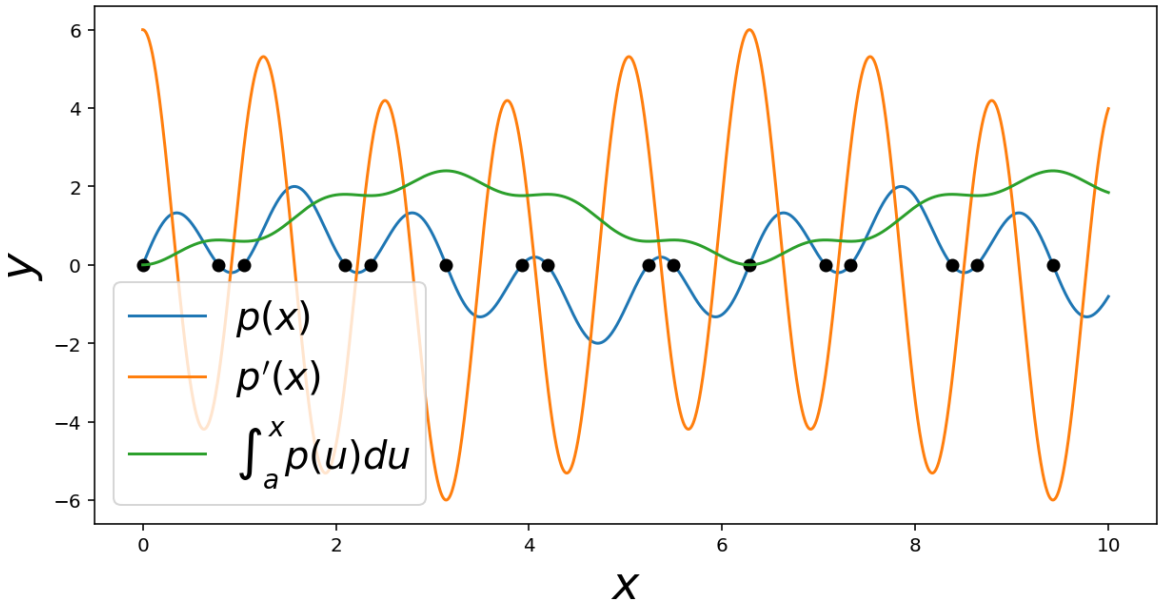
In [3]:

```
from chebpy import chebfun
x = chebfun('x', [0, 10])
f = sin(x) + sin(5*x)
fp = f.diff()
F = f.cumsum()
xroots = f.roots()

figure(1, [10, 5])
f.plot(label=r'$p(x)$')
plot(xroots, 0*xroots, 'ko')

fp.plot(label=r'$p^{\prime}(x)$')
F.plot(label=r'$\int_a^x p(u)du$')
legend(fontsize=20);
xlabel(r'$x$', fontsize=24)
ylabel(r'$y$', fontsize=24);
```

```
/anaconda3/lib/python3.7/site-packages/dask/config.py:168: YAMLLoadWarning: calling
yaml.load() without Loader=... is deprecated, as the default Loader is unsafe. Pleas
e read https://msg.pyyaml.org/load (https://msg.pyyaml.org/load) for full details.
  data = yaml.load(f.read()) or {}
```



In [ ]:

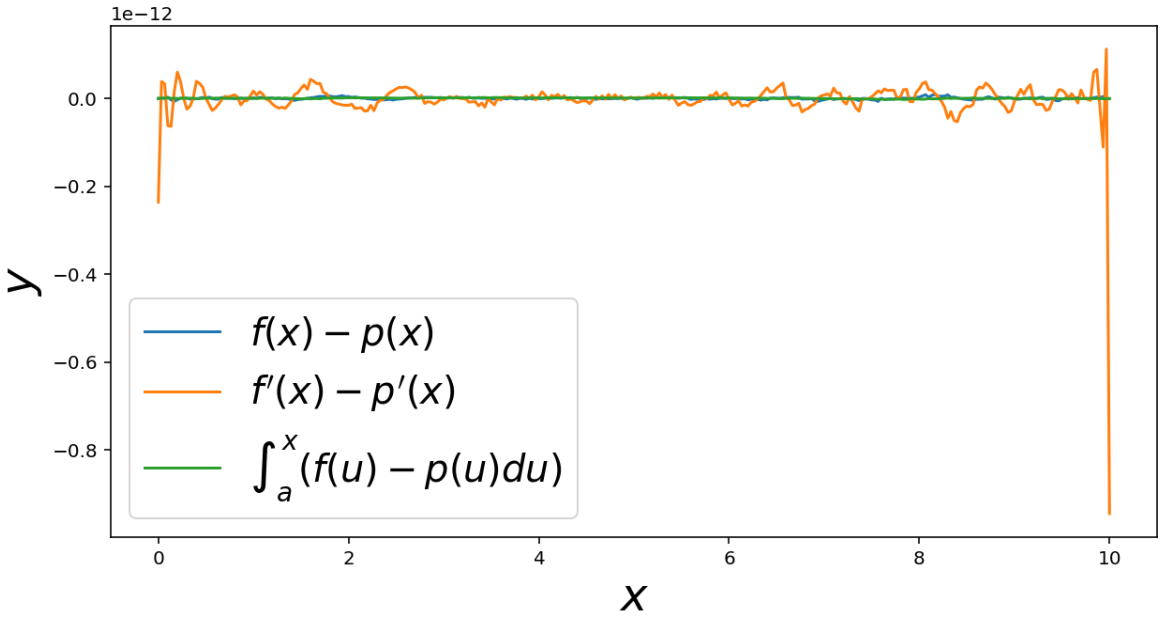
In [18]:

```
xeval = linspace(0, 10, 301)
y = f(xeval) ## f is automatically a function you can call (acts like `baryeval`)
dydx = fp(xeval)
y_integral = F(xeval)

yexact = sin(xeval) + sin(5*xeval)
dydx_exact = cos(xeval) + 5*cos(5*xeval)
y_integral_exact = -cos(xeval) - 1./5.*cos(5*xeval) - (-1. - 1./5.)

figure(1, [10, 5])
plot(xeval, yexact - y,label=r'$f(x) - p(x)$')
plot(xeval, dydx_exact - dydx,label=r'$f^\prime(x) - p^\prime(x)$')
plot(xeval, y_integral_exact - y_integral,label=r'$\int_a^x (f(u) - p(u)du)$')

legend(fontsize=20);
xlabel(r'$x$', fontsize=24)
ylabel(r'$y$', fontsize=24);
```



In [ ]:

```
# mu = 5;
# N = chebop(@(t, u) diff(u, 2) - mu*(1-u.^2).*diff(u) + u, [0 50]);
# N.lbc = [0.1; 0];
# u = N\0
# breaks = u.domain(2:end-1);
# LW = 'linewidth';
# plot(u,LW,1.2), hold on
# plot(breaks, u(breaks), 'k.', 'markersize', 14), hold off
# title('Van der Pol oscillator')
```