# 28.5: Advanced topics

This section introduces a few more advanced topics that allow you to use computer programming to simplifying many tasks. In this section, we will show you how you can write your own program to numerically estimate the value of an integral of any function.

## 28.5.1: Defining your own functions

Although Python provides many modules and functions, it is often useful to be able to define your own functions. For example, suppose that you would like to define a function that calculates $\frac{1}{2}x^2 + \frac{1}{4}x^3 + \cos(2x)$, for a given value of $x$. This is done easily using the `def` keyword in Python:

python code 28.5.1

**Defining a function**

```
#import the math module in order to use cos
import math as m

#define our function and call it myfunction:
def myfunction(x):
    return x**2 / 3 + x**3 / 4 + m.cos(2*x)

#Test our function by printing out the result of evaluating it as x = 3
print ( myfunction(3) )
```

**Output**

```
10.710170286650365
```

A few things to note about the code above:

- Functions are defined using the `def` keyword followed by the name that we choose for the function (in our case, `myfunction`)
- If functions take arguments, those are specified in parenthesis after the name of the function (in our case, we have one argument that we chose to call `x`)
- After the name of the function and the arguments, we place a colon
- The code that belongs to the function, after the colon, must be indented (this allows Python to know where the code for the function ends)
- The function can "return" a value; this is done by using the `return` keyword.
- We used the "operator" `**` to take the power of a number (`x**2`), and the operator `*`, to multiply numbers. Python would not understand something like `2x`; you need to use the multiplication operator, i.e. `2*x`.

In the example above, we wrote a Python function to represent a mathematical function. However, one can write a function to execute any set of tasks, not just to apply a mathematical function. Python functions are very useful in order to avoid having to repeatedly type the same code.

Recall that the `numpy` module allows us to apply functions to arrays of numbers, instead of a single number. We can modify the code above slightly so that, if the argument to the function, `x`, is an array, the function will gracefully return an array of numbers to which the function has been applied. This is done by simply replacing the call to the `math` version of the `cos` function by using the `numpy` version:

python code 28.5.2

**Defining a function that works on an array**

```
#import the numpy module in order to use cos to an array
```

```
02 import numpy as np
03
04 #define our function and call it myfunction:
05 def myfunction(x):
06     return x**2 / 3 +x**3 / 4 + np.cos(2*x)
07
08  #Test our function by printing out the result and evaluating it at x = 3
   (same as before)
09  print( myfunction(3) )
10
11  #Test it with an array
12  xvals = np.array([1, 2, 3])
13  print ( myfunction(xvals) )
```

**Output**

```
1 10.710170286650365
2
3 [ 0.1671865   2.67968971   10.71017029]
```

where we created the array `xvals` using the `numpy` module.

### 28.5.2: Using a loop to calculate an integral

The ability to define our own functions in Python allows us to easily simplify complex tasks. Using "loops" is another way that computer programming can greatly simplify calculations that would otherwise be very tedious. In a loop, one is able to repeat the same task many times. The example below simply prints out a statement five times:

Python code 28.5.3

**A simple loop**

```
1 #A loop to print out a statement 5 times:
2
3 for i in range(5):
4     print("The value of i is ", i)
```

**Output**

```
1 The value of i is 0
2 The value of i is 1
3 The value of i is 2
4 The value of i is 3
5 The value of i is 4
```

A few notes on the code above:

- The loop is defined by using the keywords `for ... in`
- The value after the keyword `for` is the "iterator" variable and will have a different value each time that the code inside of the loop is run (in our case, we called the variable `i` )
- The value after the keyword `in` is an array of values that the iterator will take
- The `range(N)` function returns an array of `N` integer values between `0` and `N-1` (in our case, this returns the five values $0, 1, 2, 3, 4$)
- The code to be executed at each "iteration" of the loop is preceded by a colon and indented (in the same way as the code for a function also follows a colon and is indented)

We now have all of the tools to evaluate an integral numerically. Recall that the integral of the function $f(x)$ between $x_a$ and $x_b$ is simply a sum:

$$\int_{x_a}^{x_b} f(x)dx = \lim_{\Delta x \to 0} \sum_{i=0}^{i=N-1} f(x_i)\Delta x$$

$$\Delta x = \frac{x_b - x_a}{N}$$

$$x_i = x_a + i\Delta x$$

The limit of $\Delta x \to 0$ is equivalent to the limit $N \to \infty$. Our strategy for evaluating the integral is:

1. Define a Python function for $f(x)$.
2. Create an array, `xvals`, of $N$ values of $x$ between $x_a$ and $x_b$.
3. Evaluate the function for all those values and store those into an array, `fvals`.
4. Loop over all of the values in the array `fvals`, multiply them by $\Delta x$, and sum them together.

Let's use Python to evaluate the integral of the function $f(x) = 4x^3 + 3x^2 + 5$ between $x = 1$ and $x = 5$:

Python Code 28.5.4

**Numerical integration of a function**

```
#import numpy to work with arrays:
import numpy as np

#define our function
def f(x):
    return 4*x**3 + 3*x**2 + 5

#Make N and the range of integration variables:
N - 1000
xmin = 1
xmax = 5

#create the array of values of x between xmin and xmax
xvals = np.linspace(xmin, xmax, N)

#evaluate the function at all those values of x
fvals = f(xvals)

#calculate delta x
deltax = (xmax - xmin) / N

#initialize the sum to be zero:
sum = 0

#loop over the values fvals and add them to the sum
for fi in fvals:
    sum = sum + fi*deltax

#print the result:
print("The integral between {} and {} using {} steps is {:.2f}".format(xmin,
xmax, N, sum))
```

**Output**

```
The integral between 1 and 5 using 1000 steps is 768.42
```

One can easily integrate the above function analytically and obtain the exact result of $768$. The numerical answer will approach the exact answer as we make $N$ bigger. Of course, the power of numerical integration is to use it when the function cannot be integrated analytically.

**? Exercise 28.5.1**

What value of $N$ should you use above in order to get within $0.01$ of the exact analytic answer?

**Answer**

---