

Tutorial for various functions in mathematica

```
In[22]:= Clear["Global`*"]
```

■ Basic concepts

We can represent numbers,

```
In[24]:= 2
```

```
Out[24]= 2
```

```
In[25]:= (3 + 2) / 4
```

```
Out[25]=  $\frac{5}{4}$ 
```

..., variables,

```
In[26]:= x
```

```
Out[26]= x
```

```
In[27]:= x + y
```

```
Out[27]= x + y
```

```
In[28]:= quot = (x^2 - y^2) / (x + y)
```

```
Out[28]=  $\frac{x^2 - y^2}{x + y}$ 
```

```
In[29]:= Simplify[quot]
```

```
Out[29]= x - y
```

```
In[30]:= ? Simplify
```

Simplify[expr] performs a sequence of algebraic transformations on expr, and returns the simplest form it finds. Simplify[expr, assum] does simplification using assumptions. [More...](#)

If we assign values, these are 'sticky'

```
In[31]:= x = 1
```

```
Out[31]= 1
```

```
In[32]:= quot
```

$$\text{Out}[32]= \frac{1 - y^2}{1 + y}$$

We can also store Equations in variables

```
In[33]:= eqn1 = 2 a == 3
```

```
Out[33]= 2 a == 3
```

```
In[34]:= Solve[eqn1, a]
```

$$\text{Out}[34]= \left\{ \left\{ a \rightarrow \frac{3}{2} \right\} \right\}$$

For the rest of this document it is better to 'clear' x

```
In[35]:= x
```

```
Out[35]= 1
```

```
In[36]:= Clear[x];
```

```
In[37]:= x
```

```
Out[37]= x
```

■ Functions

A simple function

```
In[38]:= func[x_] := x^2 - x^3
```

Output:

```
In[39]:= func[2]
```

```
Out[39]= -4
```

```
In[40]:= func[x]
```

$$\text{Out}[40]= x^2 - x^3$$

```
In[41]:= func[y]
```

$$\text{Out}[41]= y^2 - y^3$$

Note the 'underscore'

```
In[42]:= func2[x] := x^2 - x^3
```

```
In[43]:= func2[x]
```

$$\text{Out}[43]= x^2 - x^3$$

```
In[44]:= func2[y]
```

```
Out[44]= func2[y]
```

```
In[45]:= func2[2]
```

```
Out[45]= func2[2]
```

■ Modules

We write a simple module to return the cube of some input number

```
In[46]:= mycube[input_] :=  
  Module[{result},  
    result = input^3;  
    Return[result];  
  ]
```

```
In[47]:= mycube[3]
```

```
Out[47]= 27
```

■ 'For'-loops

We write a module returning the sum of the numbers 1,...,n (where n is the input to be specified)

```
In[48]:= mysum[n_] :=  
  Module[{result = 0, i},  
    For[i = 1, i ≤ n, i++,  
      result += i  
    ];  
    Return[result]  
  ]
```

```
In[49]:= mysum[100]
```

```
Out[49]= 5050
```

■ 'If'-conditions

The 'If' statement is not quite what it usually represents in programming languages. It's like the ternary conditional in 'c'.

```
In[50]:= If[1 == 2, 0, 100]
```

```
Out[50]= 100
```

```
In[51]:= If[2 == 2, 0, 100]
```

```
Out[51]= 0
```

Now we only want to sum the even numbers

```
In[52]:= evensum[n_] :=  
  Module[{result = 0, i},  
    For[i = 1, i ≤ n, i++,  
      result += If[Mod[i, 2] == 0, i, 0]  
    ];  
    Return[result]  
  ]
```

```
In[53]:= evensum[100]
```

```
Out[53]= 2550
```

```
In[54]:= mysum[50]
```

```
Out[54]= 1275
```

■ Creating and working with lists

We create a list of the number 0, 1, ..., 10

```
In[55]:= values = Table[x, {x, 2, 10, 1}]
```

```
Out[55]= {2, 3, 4, 5, 6, 7, 8, 9, 10}
```

We create another list with elements f[i]

```
In[56]:= flist = Table[f[i], {i, 2, 10, 1}]
```

```
Out[56]= {f[2], f[3], f[4], f[5], f[6], f[7], f[8], f[9], f[10]}
```

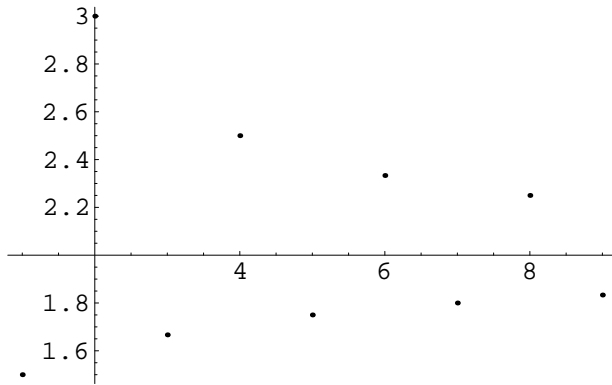
We want to assign to f[i] the value sum[i]/evensum[i].

```
In[57]:= For[i = 2, i ≤ 10, i++,  
  f[i] = mysum[i] / evensum[i]  
]
```

```
In[58]:= flist
```

```
Out[58]= { $\frac{3}{2}$ , 3,  $\frac{5}{3}$ ,  $\frac{5}{2}$ ,  $\frac{7}{4}$ ,  $\frac{7}{3}$ ,  $\frac{9}{5}$ ,  $\frac{9}{4}$ ,  $\frac{11}{6}$ }
```

```
In[59]:= ListPlot[flist]
```



```
Out[59]= - Graphics -
```

We can also directly access list elements via 'flist'

```
In[60]:= flist[[3]]
```

```
Out[60]=  $\frac{5}{3}$ 
```

```
In[61]:= For[j = 1, j ≤ 5, j++,
  flist[[j]] *= values[[j]]
]
```

```
In[62]:= flist
```

```
Out[62]= {3, 9,  $\frac{20}{3}$ ,  $\frac{25}{2}$ ,  $\frac{21}{2}$ ,  $\frac{7}{3}$ ,  $\frac{9}{5}$ ,  $\frac{9}{4}$ ,  $\frac{11}{6}$ }
```

■ Derivatives and Substituting

```
In[63]:= func[x]
```

```
Out[63]=  $x^2 - x^3$ 
```

What is func evaluated at x0? Answer:

```
In[64]:= func[x0]
```

```
Out[64]=  $x0^2 - x0^3$ 
```

```
In[65]:= func[x] /. x → x0
```

```
Out[65]=  $x0^2 - x0^3$ 
```

Both methods work fine for this example. But now we want to know the derivative f' evaluated at x+h

```
In[66]:= D[func[x], x]
```

```
Out[66]=  $2x - 3x^2$ 
```

```
In[72]:= D[func[x0], x]
```

```
Out[72]= 0
```

```
In[74]:= D[func[x], x] /. x -> x0
```

```
Out[74]= 2 x0 - 3 x0^2
```

Higher derivatives

```
In[75]:= D[x^4, {x, 3}]
```

```
Out[75]= 24 x
```

■ Solving (linear) systems of equations

We need to write the equations as a 'list', so

```
In[76]:= eq[1] = 2 x + 3 y == 4;
          eq[2] = x - 4 y == 3;
```

```
In[78]:= system = Table[eq[i], {i, 1, 2, 1}]
```

```
Out[78]= {2 x + 3 y == 4, x - 4 y == 3}
```

```
In[79]:= sol = Solve[system, {x, y}]
```

```
Out[79]= {{x -> 25/11, y -> -2/11}}
```

Note that the result is a "list of a list" of 'substitutions', not a list of numbers

```
In[80]:= sol[[1, 1]]
          sol[[1, 2]]
```

```
Out[80]= x -> 25/11
```

```
Out[81]= y -> -2/11
```

Check the result

```
In[82]:= eq[1] /. {x -> 25/11, y -> -2/11}
```

```
Out[82]= True
```

```
In[83]:= eq[2] /. sol[[1]]
```

```
Out[83]= True
```

■ Taylorexpansion

Suppose we want to expand a function $f[x]$ around a point x_0

```
In[84]:= Series[g[x], {x, x0, 3}]
```

```
Out[84]= g[x0] + g'[x0] (x - x0) +  $\frac{1}{2}$  g''[x0] (x - x0)2 +  $\frac{1}{6}$  g(3)[x0] (x - x0)3 + O[x - x0]4
```

If we want the series expansion around 'x' evaluated at 'x+h', the most straightforward idea does not work

```
In[85]:= Series[g[x+h], {x+h, x, 3}]
```

```
General::ivar : h+x is not a valid variable.
```

```
Out[85]= Series[g[h+x], {h+x, x, 3}]
```

so instead we use the substitution option

```
In[86]:= myseries = Series[g[y], {y, x, 3}] /. y -> x + h
```

```
Out[86]= g[x] + g'[x] ((h+x) - x) +  $\frac{1}{2}$  g''[x] ((h+x) - x)2 +  $\frac{1}{6}$  g(3)[x] ((h+x) - x)3 + O[(h+x) - x]4
```

We want to simplify this expressions. This requires a conversion via 'Normal'

```
In[90]:= Simplify[myseries]
```

```
Out[90]= g[x] + g'[x] ((h+x) - x) +  $\frac{1}{2}$  g''[x] ((h+x) - x)2 +  $\frac{1}{6}$  g(3)[x] ((h+x) - x)3 + O[(h+x) - x]4
```

```
In[92]:= Expand[Normal[myseries]]
```

```
Out[92]= g[x] + h g'[x] +  $\frac{1}{2}$  h2 g''[x] +  $\frac{1}{6}$  h3 g(3)[x]
```

All this is rather messy, so we use the shortcut for the expansion which is given by

$$g[x+h] + O[h]^4$$

$$g[x] + g'[x] h + \frac{1}{2} g''[x] h^2 + \frac{1}{6} g^{(3)}[x] h^3 + O[h]^4$$

$$g[x+h] + g[x-h] + O[h]^4$$

$$2 g[x] + g''[x] h^2 + O[h]^4$$

We can also access individual coefficients of the series expansion (note that the expansion parameter, the h, drops out)

```
In[94]:= mycoef = SeriesCoefficient[myseries, 2]
```

```
Out[94]=  $\frac{g''[x]}{2}$ 
```

■ Lagrange interpolation

Create a list of points

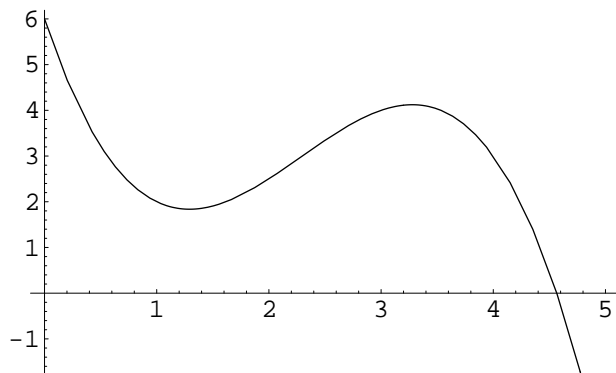
```
In[96]:= data = {{1, 2}, {2, 2.5}, {3, 4}, {4, 3}}
```

```
Out[96]= {{1, 2}, {2, 2.5}, {3, 4}, {4, 3}}
```

```
In[99]:= mypoly = Expand[InterpolatingPolynomial[data, y]]
```

```
Out[99]= 6. - 7.41667 y + 4. y2 - 0.583333 y3
```

```
In[100]:= Plot[mypoly, {y, 0, 5}]
```



```
Out[100]= - Graphics -
```