# Math 1191
# Mathematica Introduction

Lab 2

Fall, 2005

REMEMBER:

- Functions use square brackets `[]` for their arguments, *not* parentheses. Example: `Sin[3]`, `Log[10]`, `myfunction[42]`, `N[Pi,8]`
- Mathematica's built-in functions (and constants) all start with a capital letter. Example: `Sin[Pi]`, `Exp[-1]`, `Solve[3*x^2 + 4*x - 2 == 0,x]`
- It is very easy to forget which functions/variables have been defined to be what. Use the function `Clear[]` early and often so you know you are working with the right functions. Example: `Clear[x]`, `Clear[f]`.

## Defining your own Functions

Functions in Mathematica can take any sort of object as an argument and return any sort of object. For most of your initial use of functions, you will take and return numbers, but sometimes it will be useful to write functions that take (for example) a list of numbers and returns some information about that list (such as the average of the numbers, or the median of the numbers).

Let's look at an example of a function definition (in this case, of $f(x) = x^2 - 3x + 1$). In Mathematica, you would enter the following in a cell:

```
In[1]:= Clear[f, g] (* just in case f or g has already been defined *)
In[2]:= f[x_] := x^2 - 3*x + 1
```

(Whether you end the line with a semicolon or not does not matter—in either case, Mathematica does not return any output. Why is there no output?)

There are two important elements to this command. First is the use of an underscore after the variable name: `x_`. The use of an underscore is important—it tells Mathematica that this variable is going to be supplied when the function is called. (Technically, the underscore indicates a *pattern*.) Without the underscore, if you tried to evaluate `f[3]`, Mathematica would just return `f[3]` and not the actual value $f(3)$.

Another important element is the use of the delayed assignment `:=`. There is not much difference between delayed and immediate assignment, but in general it is better to use

the delayed assignment. This prevents Mathematica from trying to evaluate the function definition before you really want it to (which can avoid some errors)

The rest of the function definition is what you would probably expect: the expression $x^2 - 3x + 1$ translated into Mathematica.

When the function is called with `f[3]`, Mathematica simply replaces every instance of `x` in the function definition with a `3` and evaluates the expression, returning the result. Most of the time this will be a number, but you can also use it to compose functions:

```
In[3]:= g[x_] := 4 + x
In[4]:= f[3]
Out[4]:= 1
In[5]:= f[g[x]]
Out[5]:= 1 - 3(4 + x) + (4 + x)^2
In[6]:= Simplify[f[g[x]]]
Out[6]:= 5 + 5 x + x^2
```

Consider the following commands and output:

```
In[7]:= x = 4;
In[8]:= f[x]
Out[8]:= 5
```

That is probably not what we expected for `f[x]`. The problem is *not* that Mathematica has somehow changed the definition of $f$. The problem is that once `x` has the value of `4`, the call to `f[x]` is the same as calling `f[4]`. This can be annoying if you use common variable names in your function definitions, set one of those variables to a value outside the function definition, then decide you want to know the definition of `f[x]`.

However, you can use the function `Definition[]` to see how a function is defined:

```
In[9]:= Definition[f]
Out[9]:= f[x_] := x^2 - 3 x + 1
```

Defining a function that takes more than one variable as an argument is done in a similar way. The following function gives the distance between the given point $(x, y)$ and $(2, 1)$:

```
In[10]:= d[x_,y_] := Sqrt[(x-2)^2 + (y - 1)^2]
In[11]:= d[2,3]
Out[11]:= 2
```

It is also possible to define default values for arguments, using a colon. For example, the following function will give the distance between two points $(x, y)$ and $(a, b)$, but if the second point is not given then the default is the origin $(0, 0)$.

```
In[12]:= Clear[d];
In[13]:= d[x_,y_,a_:0,b_:0]:= Sqrt[(x-a)^2 + (y-b)^2]
In[14]:= d[2,3]
Out[14]:= √13
In[15]:= d[2,3,2,1]
Out[15]:= 2
```

*A Note on Function Names.* You can name your function however you wish, as long as Mathematica does not already have a function (or constant or other object) named the same. It is a good practice to avoid function names like `f[]` or `g[]` for anything other than quick and dirty calculations—instead, give your functions descriptive names so you will know what they are supposed to do.

At times, you may give a name to a function (or variable) that is *similar* to a name already in use. In this case, Mathematica will beep at you and display a warning message. Don't worry: your function has been defined with the name you gave it. But you might consider using a different name so that there is less chance for confusion.

## Functions with Non-Numeric Arguments and Values

Take a look at the last example (finding distance between two points with a default value). The use of the function is not especially nice—we have to give it 4 arguments when, conceptually, we should be giving it only 2 (the two points). You should write functions that are intuitively easy to use, especially since you will almost immediately forget how to use them (trust me, you will).

So let's rewrite the distance function but this time so that we can pass a point to it. A point will be represented as a list (remember those from Lab 1?), such as {2,3} or {0,0}. We will use a capital letter to denote a point. The following function finds the distance between the points $P$ and $Q$:

```
In[16]:= Clear[d];
In[17]:= d[P_,Q_]:= Sqrt[ (P[[1]] - Q[[1]])^2 + (P[[2]] - Q[[2]])^2]
In[18]:= d[2,3,2,1]
Out[18]:= 2
In[19]:= d[1,1,0,0]
Out[19]:= √2
```

And we can still set default values:

```
In[20]:= Clear[d];
In[21]:= d[P_,Q_:{0,0}] := Sqrt[ (P[[1]] - Q[[1]])^2 + (P[[2]] - Q[[2]])^2]
In[22]:= d[2,3]
Out[22]:= √13
```

Lists will be a common argument for functions. Here's an example that finds the range of a list of numbers (the largest minus the smallest):

```
In[23]:= rng[L_]:= Max[L] - Min[L]
In[24]:= rng[{4,-1,5,-3,-6,7,3,1,3,8}]
Out[24]:= 14
```

## Piecewise Defined Functions

Recall that a piecewise defined function has different definitions on different parts of its domain. For example, a function that takes the square root of any positive number but squares any non-positive number would be defined as

$$f(x) = \begin{cases} \sqrt{x} & \text{if } x > 0 \\ x^2 & \text{if } x \leq 0 \end{cases}$$

These kinds of functions can be defined in Mathematica in one of two ways: using `Which[]` or using argument restriction. We will look at both ways.

Using `Which[]` is not difficult. The argument to the `Which[]` function is a list of tests and values: Mathematica checks each test (starting on the left) and when it finds a test that comes out *true*, it returns the next value. For example:

```
In[25]:= Clear[f];
In[26]:= f[x_] := Which[x <= 0, x^2, x > 0, Sqrt[x]]
In[27]:= f[2]
Out[27]:= √2
In[28]:= f[-5]
Out[28]:= 25
```

We can also force `Which[]` to output a default value by making the last test `True`:

```
In[29]:= Clear[g]
In[30]:= g[n_] := Which[n==2, "it's two!", n==3, "it's three!", True, "not two
In[31]:= g[2]
Out[31]:= it's two!
In[32]:= g[5]
Out[32]:= not two or three
```

The previous example also shows that the output can be strings (enclosed by double quotes) in addition to numbers.

Argument restriction is similar to `Which[]`, but the tests on the input are done as part of the argument definition with `/;`.

```
In[33]:= myln[x_ /; x > 0] := Log[x]
 myln[x_  x <= 0] := "Invalid input for logarithm"
In[34]:= myln[10]
Out[34]:= Log[10]
In[35]:= myln[-2]
Out[35]:= Invalid input for logarithm
```

Either form will work for piecewise defined functions—it is possible to translate one into the other. In general, using `Which[]` is preferable.

# Assignment

1. If you deposit $P$ dollars in an account that is compounded continuously at $100r\%$ annual interest rate, then after $t$ years the amount $A$ that you will have in the account is given by $A = Pe^{rt}$.

   (a) Write a function named `ContComp[]` that computes $A$ given $P$, $r$ and $t$.

   (b) Use your function to compute the amount in the account if \$2,500 is deposited at 5% for 4 years.

   (c) Use your function with the `NSolve[]` function to determine how many years it will take an initial deposit of \$500 to grow to \$1000 at an annual interest rate of 5%.

2. The volume of a right circular cone of height $h$ and radius $r$ is $\frac{1}{3}\pi r^2 h$. The lateral surface area is $\pi r\sqrt{r^2+h^2}$.

   (a) Write a function called `Cone[]` that takes as input the radius $r$ and height $h$ of a right circular cone and returns *as a list* the cone's volume and lateral surface area.

   (b) Use your function to find the volume and lateral surface area of a cone with height 4 and radius 2.

3. Write a function called `slope[]` that will find the slope between two points. The points should be represented as lists of numbers (so the function will take two arguments, one for each point). Check your function by finding the slopes between some pairs of points.

4. In the text above, we used `Which[]` to define the function

$$g(n) = \begin{cases} \text{It's two!} & \text{if } n = 2 \\ \text{It's three!} & \text{if } n = 3 \\ \text{not two or three} & \text{otherwise} \end{cases}$$

   and argument restrict to define the function

$$myln(x) = \begin{cases} \ln(x) & \text{if } x > 0 \\ \text{Invalid input for logarithm} & \text{if } x \leq 0 \end{cases}$$

   Write the function $g$ using argument restriction and the function $myln$ using `Which[]`.