Milestone 2 Demo: Group 11

To make it more interactive to understand how to use our package, we provide this demo part to show how our module works. Every cell can be run by yourself and feel free to change the function to your own interest!

First of all, we need to import our packages:

In [1]:
```
from AutoDiff import ad
from BasicMath import *
```

Then, we can make use of the `auto_diff` method in our `ad` class to compute the nominal value and the derivative of the function you care about at a specific evaluation point and order.

`auto_diff` method has 3 augments:

- (1) function: the function of your interest.
- (2) eval_point: the point at which you want to compute at.
- (3) order: the order of the derivative you want to compute and its default is 1; unfortunately, our package only supports the first order derivative for now.

Let's define our demo function first!

(1) Define the function:

We can start from a simple function:

$$f(x) = x^3 + 1$$

There are many ways for you to define the function.

For example, you can either write your own function like as below:

In [2]:
```
def func(x):
    return x**3 + 1
```

Or you can write it as a lambda function like this:

In [3]:
```
func = lambda x: x**3 + 1
```

Both work perfectly in our packages.

(2) Define the evaluation point:

Then you have to define the point you want to evaluate at.

For example, we want to choose $x = 1$ as our evaluation point and we can use the code below:

In [4]:
```
x = 1
```

(3) Define the order:

For now we can only support the first order derivative. Since the default for this is 1, there is no need for you to further define it. But for future use, as our package is going to have high order differentiation, you can define the order to your own interest.

You can get the value and derivative for our function now!

In [5]:
```
t = ad().auto_diff(function = func, eval_point = x, order = 1)
```

t here is a DualNumber class object. You can see the nominal value of the function by calling the attribute `val` and see the derivative by calling the attribute `der` .

In [6]:
```
print('The nominal value of our function is {}'.format(t.val))
print('The derivative is {}'.format(t.der))
```

```
The nominal value of our function is 2
The derivative is 3
```

Now it's your time to change the function to whatever you want and get the value and derivative you want using the complete code below:

In [7]:
```
# you can change it to your function below, we just provide a more complex demo here
your_func = lambda x: x**x + 1/2*log(x) + (-x)

# choose your evaluation point
your_x = 1

your_t = ad().auto_diff(your_func, your_x)

print('The nominal value of your function is {}'.format(your_t.val))
print('The derivative is {}'.format(your_t.der))
```

```
The nominal value of your function is 0.0
The derivative is 0.5
```