


# Interpreter - call the relevant function

```
def process_ast_exp(ast: ast_types.ASTNode):  
    result = None  
    match type(ast):  Uses pattern matching, introduced in Python 3.10  
        case ast_types.Number:  
            result = process_number(ast)  
        case ast_types.NegExp:  
            result = process_neg_exp(ast)  
        case ast_types.AddExp:  
            result = process_add_exp(ast)  
        case ast_types.SubExp:  
            result = process_sub_exp(ast)  
        case ast_types.MulExp:  
            result = process_mul_exp(ast)  
        case ast_types.DivExp:  
            result = process_div_exp(ast)  
    if result is None:  
        raise SyntaxError("Unable to process AST of type {}".format(type(ast)))  
    return result
```

# Interpreter - evaluating Numbers

General idea: process the stuff in the "data" (or "left" and "right", if a binary expression like  $x + y$  or  $x * y$ ) attributes of the AST node, and return the result.

If unable to process the attached node data, then raise an error.

Number is the base case for all our recursive calls, so it doesn't call any other interpreter functions.

Uses type hinting, introduced in Python 3.5



```
def process_number(ast: ast_types.Number):  
    try:  
        return int(ast.data)  
    except:  
        raise SyntaxError('Unable to process "{}" as number'.format(ast.data))
```