第 4 节：GEE 的数据类型（String, Number)

String 和 Number 是 GEE 中最基本的数据类型。这两个变量的中文含义是“文本”和“数字”。无论利用代码进行哪种操作，都免不了要利用文本和数字对操作进行表述或者控制。本节的目的就是要介绍这两种数据类型在 GEE 中的基本用法，通过本节的学习，我们能够掌握这两种基本数据类型的构建函数，以及理解 GEE 命令语言的基本语法。

4.1 String

文本本身不具备数据功能，即它通常不参与运算，而只是用于描述。因此我们首先学习如何将文本打印在 Console 栏中，具体命令如下：

```
print ('this is a string')
```

该命令可以从以下角度进行理解。这段代码里涉及到两个部分，第一个部分是 GEE 自带的命令 print()，该命令的功能是将括号内的内容输出到 Console 栏中；第二个部分是括号内的内容 'this is a string' 。

那么问题来了，为什么不能直接把 this is a string 放在括号内，而非要加一个单引号呢？答案是英文字母或者单词在代码中通常被用作变量名(类似我们在初高中数学里经常用到的那种“假设小明有 a 元，小红有 b 元”中的 ab 变量)，如果不对文本加以处理，那么 GEE 看到 this is a string 就有可能晕掉，因为它不能分辨出里边是否有变量名。因此，为了减少误解，必须对文本进行一定的处理，在代码中，这种避免误解的默认处理方式就是在文本两端加上单引号(' ')或者双引号(" ")。

对文本进行处理之后，我们尝试用 GEE 的思维来看这一段代码：首先，print()是打印的功能，哦，懂了，我要把括号里的内容打印到 Console 里去。然后括号里是'this is a string'，哦，明白，引号中间的是文本，所以我要打印的内容是 this is a string。

最后，代码的效果如下图所示。

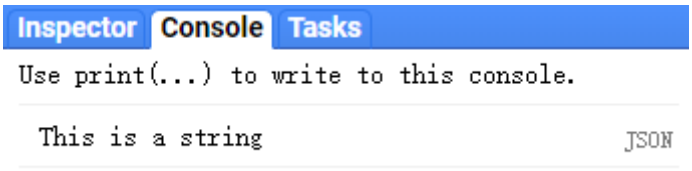

图 4.1 文本的打印

下边，我们探索如何创建一个 String 变量。具体代码如下：

```
var string = ee.String('this is a string')

print(string )
```

在这段代码中，我们接触到了两种新的命令方式，即 **var** string 和 ee.String('this is a string')，其中 ee.String('this is a string')可以用前边的思路来理解，即 ee.String( )告诉 GEE 这是一个文本，并且具体的文本内容在括号内。而括号内的'this is a string'可以用上一个例子来解释。另一个新内容是 **var** string = ，对于这部分指令，可以这样理解，即 **var** 告诉 GEE 我们要创建变量了，变量的名字跟在 **var** 后边；然后是等号 =，等号的含义在于告诉 GEE 这个变量名等于等号后边的内容。

综上，我们用 GEE 的思路来理解一下 **var** string = ee.String('this is a string')。首先我要创建一个名为 string 的变量，string 的值等于 ee.String('this is a string')，也就是说，string 首先等于一个文本，然后文本的具体内容是 this is a string。在往下看，要执行 print(string)，哦，明白了，就是要把括号里的内容打印到 Console 里，括号里是什么呢？是一个名为 string 的变量，因为 string 是变量，所以我要打印 string 代表的内容，也就是打印 this is a string。

在这里，需要进一步解释两点。第一点，**var** string = ee.String('this is a string')的正确读法应该是从右往左，即有一个值为 this is a string 的文本被赋值给了名为 string 的变量。第二点，下述代码与上述代码是完全等效的，但却是不推荐的代码编写方法。因为在给变量赋值的时候，我们不仅关注值是什么，而且还关注值是什么格式。为什么多此一举非要使用 ee.String( )的命令呢？这是因为随着代码的增多，我们在检查代码的时候通常会把注意力较多的放在代码的逻辑上，但这种数据格式的不规范有时会导致错误，有时不会导致错误，属于较为隐蔽的错误。所以在代码刚开始编写的时候，就要特别注意对变量的数据格式进行定义。这种定义数据格式的行为在 Java 代码中被称为“cast”。

```
var string = 'this is a string'

print(string )
```

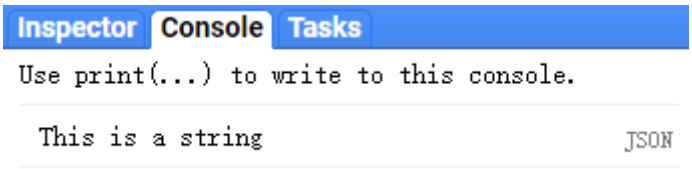

图 4.2 文本变量的打印

在做上述讲解过后，我们应该对 GEE 的命令格式和编写规范有了一定的认识，下边开始介绍 String 的合并命令，具体代码如下：

```
var string_1 = ee.String('I am the first.')
var string_2 = ee.String('I am the second.')
var cat_string = string_1.cat(string_2)

print (cat_string)
```

在这个代码中，我们遇到了新的指令：.cat()，这个指令的含义在于将.(点)前边的文本和 cat 后边的括号里边的文本进行合并。需要指出的是，cat 命令的英文单词对应的是“catenate”，含义为“连接，耦合”。代码的执行结果如下图所示。

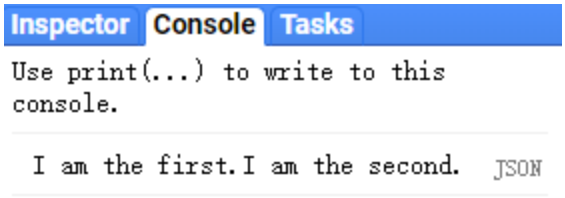

图 4.3 文本的合并

下边我们学习文本的替换命令，具体代码如下：

```
var string_1 = ee.String( "ABC ADE AFG AHI" );
var string_2= string_1.replace( 'A', '-' );

print( string_1, string_2);
```

在代码中，我们接触到的新命令是.replace( )，这个命令一共有两个参数，分别在括号内以第一个和第二个位置进行表示。那么这个命令的具体含义就是，对.(点)之前的文本进行替换操作，替换的方法是把原来文本中第一个包含文本 1 的内容替换为文本 2 的内容。具体结果如下：

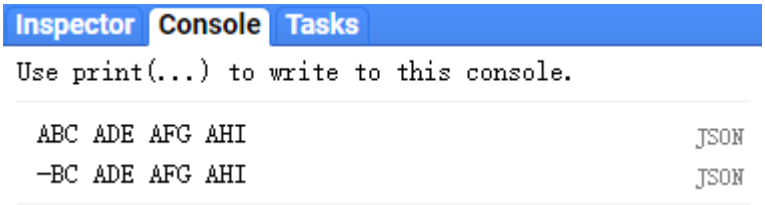

图 4.4 文本的替换

下边我们学习文本的匹配命令，具体代码如下：

```
var string_1 = ee.String( "A B C D C" );
var string_2= string_1.match( 'C');
print( string_1, string_2);
```

这里新的命令式.match('C')，我们可以通过前述的例子进行理解，具体结果如下图：

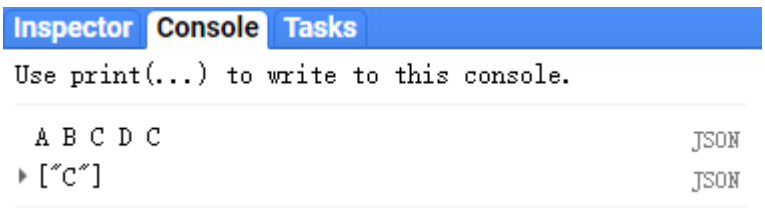

图 4.5 文本的匹配

下边我们学习文本的分割，具体代码如下：

```
var string_1 = ee.String( "A_B_C_D" );
var string_2= string_1.split( '_' );

print( string_1, string_2);
```

在这个指令中，.split('_')把前边文本根据括号内的文本(或者符号)进行分割后得到新的 List 数据(下一节会学习)。代码效果如下：

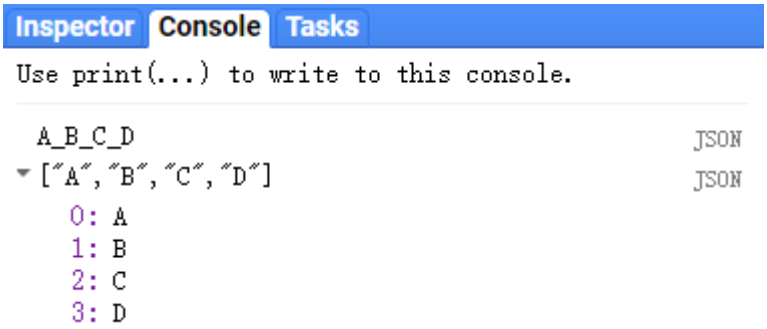

图 4.6 文本的切割

下边我们学习文本的截取，具体代码如下：

```
var string_1 = ee.String( "1234 5678 90AB" );
var string_2= string_1.slice( 5, 10 );
var string_3= string_1.slice( 5);
var string_4= string_1.slice( -5);

print( string_1, string_2, string_3, string_4);
```

在这个命令中，.slice(,)的功能是将前边的文本取出来一部分，具体的取法根据括号内的数字来确定，如果是两个数字，那么取从第一个数字(不包含)开始到第二个数字(包含)截止的文本部分；如果只有一个数字，就从这个数字开始(不包含)取到文本结尾。假如这个数字是负数，那么开始的位置就是从右往左数的。具体代码效果如下：

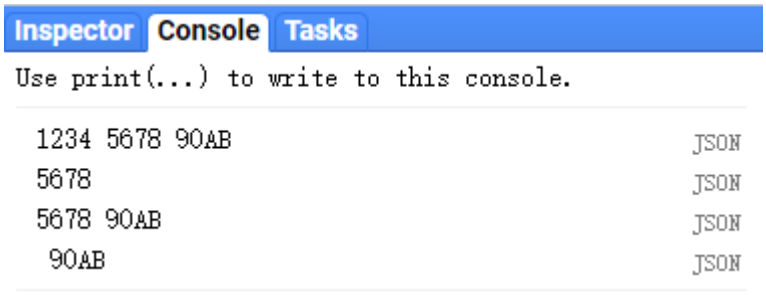

图 4.7 文本的截取

下边介绍文本的长度测量命令，具体代码如下：

```
var string = ee.String( 'ABCD 1234' );
var number = string .length( );

print(string , number );
```

这个命令的功能在于计算给定文本的长度。具体代码结果如下图：

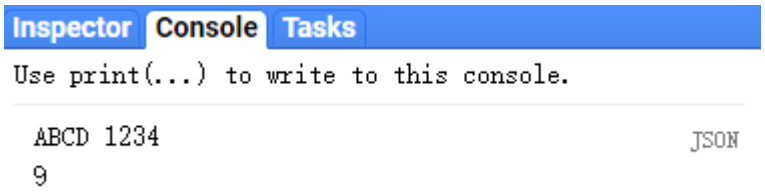

图 4.8 文本的长度计算

经过本小节的学习，我们初步掌握了 GEE 的基本命令格式以及文本的基础操作。下边给出上述所有命令的集合，尝试能否看到指令就联想到其功能与用法。

print()	ee.String()	string.cat()	string.replace()
string.split()	string.match	string.slice()	string.length()

4.2 Number

Number 是 GEE 中另一个最基础的数据类型，本节我们将学习 Number 的常用命令。

首先是创建数字，其代码和结果如下：

```
var number= ee.Number( 1324567980 );

print(number);
```

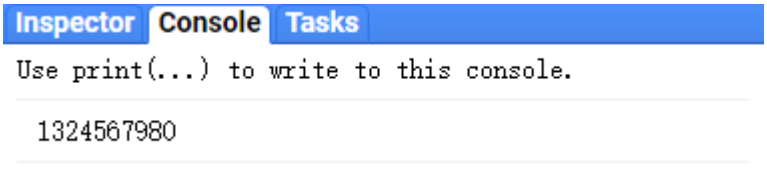

图 4.9 数字的创建

下边是数字格式的转换，代码和执行效果如下

```
var number_1 = ee.Number( -3.14159 );
var number_2 = number_1.int8();
var number_3 = number_1.toInt8();

print( number_1, number_2, number_3 );
```

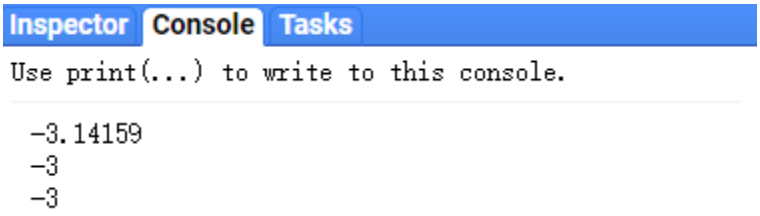

图 4.10 数字的格式转换

在这里需要注意两点，第一是 GEE 中数字在转换格式的时候.int8()与.toInt8()是等效的；第二点是数字在转换格式的时候可能会发生数据丢失，这就要求我们在处理数字的时候，不仅要关注数字本身，还要留意相应的处理是否会改变原来数字的格式范围。另外，在 GEE 中其它常见的数字转换命令如下：

.unit8/16/32/64	=	.toUnit 8/16/32/64
.float	=	.toFloat
.double	=	.toDouble

下边介绍数字的比较，代码和运行结果如下：

```
var Nuber_1 = ee.Number(10);
var Nuber_2 = ee.Number(-10);
var True_False    = Nuber_1.eq( Nuber_2 );

print( Nuber_1, Nuber_2 )
print( True_False);
```

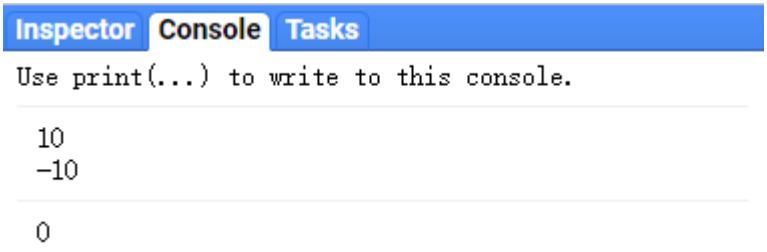

图 4.11 数字的比较

需要注意的是，在代码通常用数字 1 来表示条件成立(真)，用数字 0 来表示条件不成立(假)。同样的，在 GEE 中其它常见的数字比较命令如下：

number.eq()	number.neq()	number.gt()	number.gte()	number.lt()	number.lte()
=	≠	>	≥	<	≤

另外，两个数字之间还可以进行逻辑操作，限于篇幅关系这里不给代码，只给出命令以方便对 GEE 的学习。

number.and()	number.or()	number.not()

下边介绍数字的函数运算，代码及运行效果如下：

```
var number_1 = ee.Number( -3.1415);
var number_2 = number_1.floor().abs();

print ( number_1, number_2 );
```

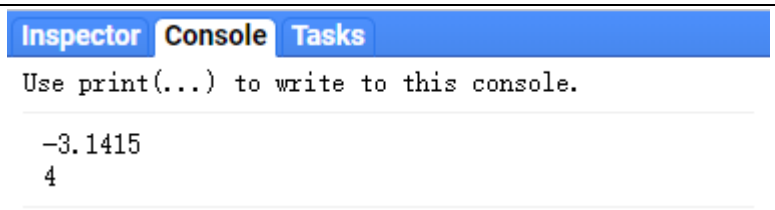

图 4.12 数字的函数操作

在这个例子中，我们接触到的命令是.floor()和.abs()，他们分别代表向下取整(即比它小的最大整数)和取绝对值。GEE 中其它常见的函数操作如下所示：

number.round()	number.ceil()	number.sqrt()	number.exp()	number.log()	number.log10()
四舍五入	向上取整	开方	幂	对数	10 底对数

下边介绍数字的数学运算，代码及运行结果如下：

```
var number_1 = ee.Number( 3.14 );
var number_2 = ee.Number( 1.41 );
var result = number_2.subtract(number_1);

print( number_1, number_2 );
print( result );
```

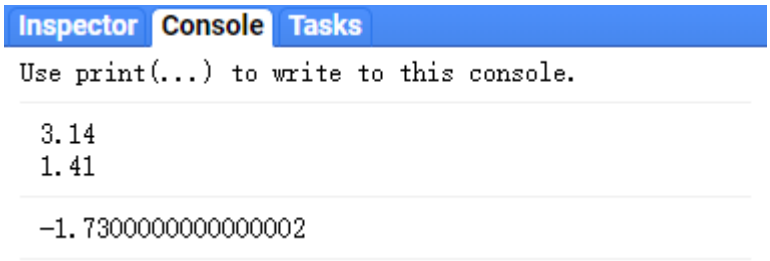

图 4.13 数字的数学操作

在这个例子中，.subtract() 代表用前边的数字减去后边的数字的含义，但在结果中出现了许多 0，并且在最后出现了一个 2，这是由于计算机在计算的时候内存中没有完全初始化造成的，这种误差一般不会影响最终结果。同样的，其它 GEE 重点数学运算如下：

number.add()	number.multiply()	number.divide()	umber.max()	number.min()
加	乘	除	最大值	最小值
number.mod()	number.hypot()	number.first()	number.first_nonzero()	
取模	算三角形斜边	取第一个	取非零第一个	

下边介绍数字的三角函数，代码及运行效果如下：

```
var Degree    = 45;
var Radian    = ee.Number( Degree / 180 * 3.1415926 );
var Tangent = Radian.tan();

print( Degree );
print( Radian );
print( Tangent );
```

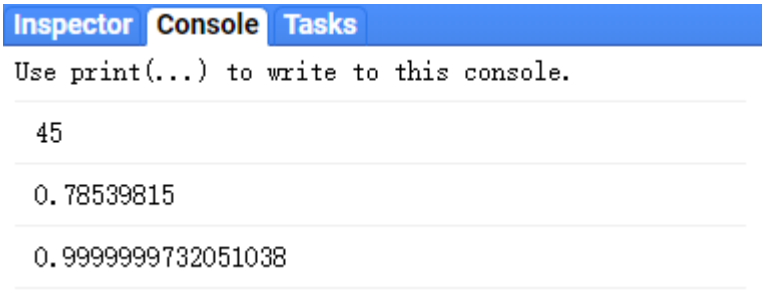

图 4.14 三角函数操作

本例中.tan()的作用是求取前边数字(弧度形式)的正切值，GEE 中其它三角函数如下所示：

.sin	.cos	.sinh	.cosh	.tanh	.acos	.asin	.atan
sin	cos	双曲正弦	双曲余弦	双曲正切	反余弦	反正弦	反正切

下边介绍数字的是非比较，代码及运行效果如图所示：

```
var number_1 = ee.Number( 3.14 );
var number_2 = ee.Number( 3.14 );
var number_3 = ee.Number( 3.141);
var True_false_1    = ee.Algorithms.IsEqual( number_1 ,number_2 );
var True_false_2    = ee.Algorithms.IsEqual( number_2, number_3 );

print( number_1, number_2,number_3 );
print( True_false_1, True_false_2 );
```

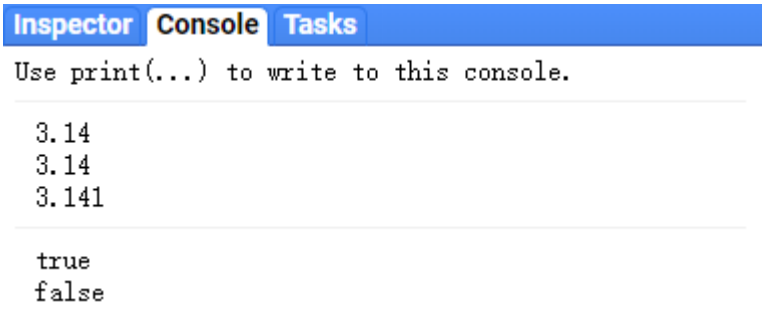

图 4.15 数字的是非比较

这里 ee.Algorithms.IsEqual()的功能是比较括号内的两个数字是否相同，如果相同的话返回一个文本 true，如果不同的话返回一个文本 false。

下边介绍数字的位运算，代码及运行效果如下：

```
var Number_1   = ee.Number( 1 );
var Number_2   = ee.Number( 2 );
var Number_And = Number_1.bitwiseAnd( Number_2 );
var Number_Or  = Number_1.bitwise_or( Number_2 );

print ( '00000001 (=1) and 00000010 (=2)', Number_And  );
print ( '00000001 (=1)  or  00000010 (=2) ', Number_Or );
```

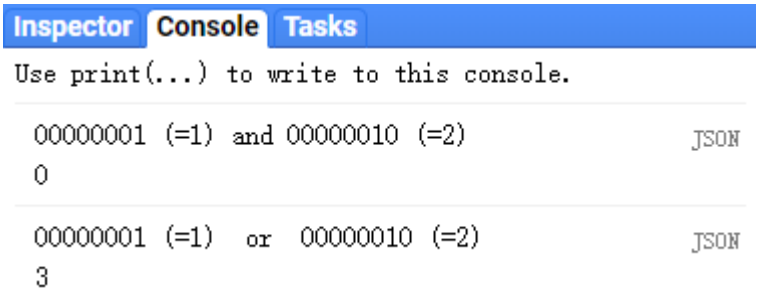

图 4.16 数字的与或位运算

在这里需要注意的是，位运算就是将数字首先转换成二进制形式，然后对相对应位置的两个数字进行比较。具体到这个例子，1 变成二进制的形式是 00000001(因为 1 是 int8 格式，所以转变成二进制后一共 8 位)，2 变成二进制形式是 00000010，把两个二进制形式的数字的对应位置进行 and 操作，就是把 00000001 / 00000010 中上下两个对应位置进行 and 比较，最终得到的二进制结果是 00000000，转换成十进制数字就是 0，所以输出结果是是 0。类似的，对上下两个位置进行 or 操作，最终得到的二进制结果是 00000011，转换成十进制数字就是 3，所以输出的结果是 3。

另外，GEE 中类似的位运算命令如下：

bitwise.And/Or/Xor/Not	=	bitwise_and/_or/_xor/_not
与/或/异或/非		与/或/异或/非

下边介绍位运算的移位操作，具体代码和运行效果如下：

```
var number   = ee.Number( 3 );
var number_left  = number.leftShift( 2 );
var number_right = number.rightShift( 1 );

print( '00000011 to 00001100',  number_left  );
print( '00000011 to 00000001',  number_right );
```

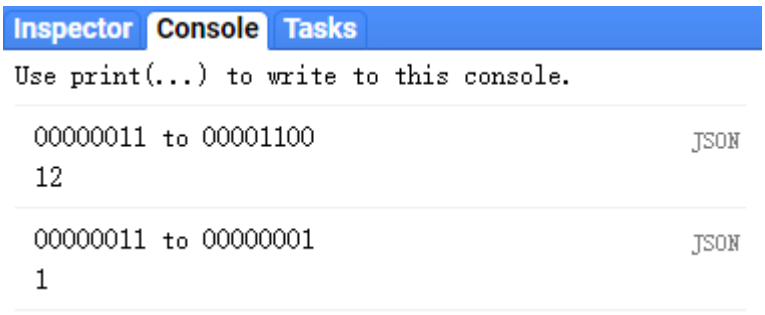

图 4.17 数字的移位位运算

移位位运算的功能是将数字变成二进制后，将二进制中所有的数字向左或者向右移动一定的距离，产生的空档用 0 补充。本例中 3 的二进制形式是 00000011，向左移动两个位置后得到新的二进制数字是 00001100，其对应的十进制数字是 12。这里指出，二进制数字向左移动一位，相当于原十进制数字乘以 2，移动两位相当于原十进制数字乘以 2 后再乘以 2，大家可以思考一下为什么会这样。同样的，向右移动相当于除以 2，但本例中的原十进制数字 3 向右移一位以后却变成了 1，建议大家自己画一下，尝试解释这种现象的原因。

下边是本节有关 Number 的主要命令，大家尝试能否看到命令就回想到相应的格式与功能。

ee.Number()	number.uint8()	number.toUint8()	number.int8()	number.toInt8()
number.eq()	number.neq()	number.and()	number.or()	ee.Algorithms.IsEqua()
number.abs()	number.round()	number.pow()	number.bitwise_and()	
number.bitwise_or()		number.leftShift()	number.right_shift()	

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

本材料由王金柱（西南大学&迪肯大学）创作。如有需要请与我联系。
This doc contributed by Jinzhu Wang of Southwest University & Deakin University.
Email: wangjinzhulala@gmail.com