```
In[•]:= f_x = g_x * Exp[I * (x + y)]
Out[•]=
        e^(i (x+y)) g_x

In[•]:= D[f_x, x]
Out[•]=
        i e^(i (x+y)) g_x + e^(i (x+y)) Subscript^(0,1)[g, x]

In[•]:= Needs["Notation`"]

        Symbolize[ f_x ] (*通过回车隔开是可行的*)

        Symbolize[ g_x ]

In[•]:= f_x = g_x * Exp[I * (x + y)]
Out[•]=
        e^(i (x+y)) g_x

In[•]:= D[f_x, x]
Out[•]=
        i e^(i (x+y)) g_x
```

目标：创建包含小写下标的符号，并对其正确求导

正确的做法

可行的做法：使用 Notation`Symbolize 函数，并注意: 使用 Symbolize 函数时，**不要和其它非 Symbolize 的表达式通过分号混合使用**

注: 运行前, 先鼠标点击面板 Evaluation -> Quit kernel -> local. 相当于关闭内核，下次运行时就是新的内核

```
In[•]:= Needs["Notation`"]

In[•]:= Symbolize[ u_x ]; Symbolize[ u_x1 ] (*注意，这里是回车，不是分号*)
        u_x = u_x1[x] * Exp[I * (x + y)]
Out[•]=
        e^(i (x+y)) u_x1[x]

In[•]:= D[u_x, x]
Out[•]=
        i e^(i (x+y)) u_x1[x] + e^(i (x+y)) u_x1'[x]
```

错误的做法

错误的做法：**非 Symbolize 的表达式通过分号与 Symbolize函数混合使用**
注: 运行前, 先鼠标点击面板 Evaluation -> Quit kernel -> local.

```
In[•]:= Needs["Notation`"]

In[•]:= Symbolize[ u_x ]; Symbolize[ u_x1 ];
        u_x = u_x1 * Exp[I * (x + y)] (*注意，这里通过分号将 Symbolize 语句和 u_x 的赋值语句合在一起了 *)
Out[•]=
        e^(i (x+y)) u_x1

In[•]:= D[u_x, x]
Out[•]=
        0
```

详细解释错误原因

1. 首先，需要了解 MMA 处理表达式的流程:
A. 前端输入的表达式为 box structure (MakeExpression ,TagBox 等函数大致在此发挥作用。box structure 可以通过面板 Cell -> show expression 查看)
B. 这些 box structure 被解析 (parse) 为可以被内核执行的 expression (对应于 FullForm, FullForm大致就是内核要计算的表达式,
自定义的 UpValues, OwnValues, DownValues 可以影响expression的形式和后续的evaluation process)
C. 后端计算 (Evaluation, 就像在执行 *expression //. {all global rules}*)
D. 将计算结果转换为 box structure 输出到前端 (MakeBoxes 等函数大致在此发挥作用)

前端输入(*A*) $\xrightarrow{\text{解析box structure}}$ **FullForm的expression(*B*)** $\xrightarrow{\text{后端计算, 类似于 } expression\ //.\ \{all\ global\ rules\}}$
得到计算结果(*C*) $\xrightarrow{\text{转换为可显示的box}}$ **前端显示box structure(*D*)**

2. **Symbolize**[□] 函数会在 (A->B) box structure 被解析的这个过程中发挥作用

3. 分号 ; 在 MMA 中其实是 CompoundExpression[expr1, expr2, ...] 这个函数，相当于 expr1; expr2; ... 该函数将多个子表达式合并为一个大表达式，并且抑制分号前面表达式的输出 (输出为 Null)
CompoundExpression 这个函数**只限定了(C) evaluation 的顺序**是先 expr1, 后 expr2。**并没有限定 (A->B) 解析的顺序**
因此，如果将 Symbolize 函数与其它语句通过分号隔开，并不会按我们预想的顺序去解析

因此，**使用分号 ; 时，应该警惕语句中是否有关于 box structure 解析过程处理的语句**
参考:
https://reference.wolfram.com/language/Notation/tutorial/ComplexPatternsAndAdvancedFeatures.html

注: 运行前, 先鼠标点击面板 Evaluation -> Quit kernel -> local.

```
In[•]:= Needs["Notation`"]
```

In[]:=
```
Symbolize[ f_y ]

FullForm[f_y] (* 可见，Symbolize 处理过的符号不再是 Subscript[f,y]，
 而是 f\[UnderBracket]Subscript\[UnderBracket]y *)
```

Out[]//FullForm=
```
f\[UnderBracket]Subscript\[UnderBracket]y
```

In[]:=
```
Symbolize[ u_x ]; Symbolize[ u_x1 ]; u_x = u_x1 * Exp[I * (x + y)]
```

Out[]=

$$e^{i\,(x+y)}\,u_{x1}$$

```
FullForm[u_x] (*通过执行上一个命令，
 发现 u_x 已经被正确转换为别的形式，而不是表示为 Subscript[u,x]
  但是因为赋值语句和 Symbolize 语句的解析过程是同时的，
 因此 u_x1*Exp[I*(x+y)] 这个表达式被赋值给了 Subscript[u,x]，
 而不是 u\[UnderBracket]Subscript\[UnderBracket]x
 *)
```

Out[]//FullForm=
```
u\[UnderBracket]Subscript\[UnderBracket]x
```