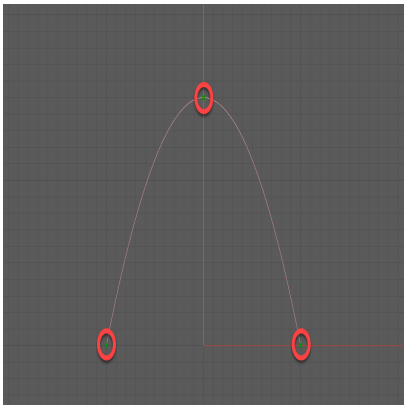


3 Point Parabola in Grasshopper + Python

Description

A Parabola based on 3 given Points can be made with Grasshopper – but you'll have to script it. Here, I did it with Python: not as difficult as it seems!

The other day I was stuck with a seemingly simple task: to produce a parabola in Grasshopper, based on 3 given Points. Something like the 3-Point-Circle you find in every CAAD software.

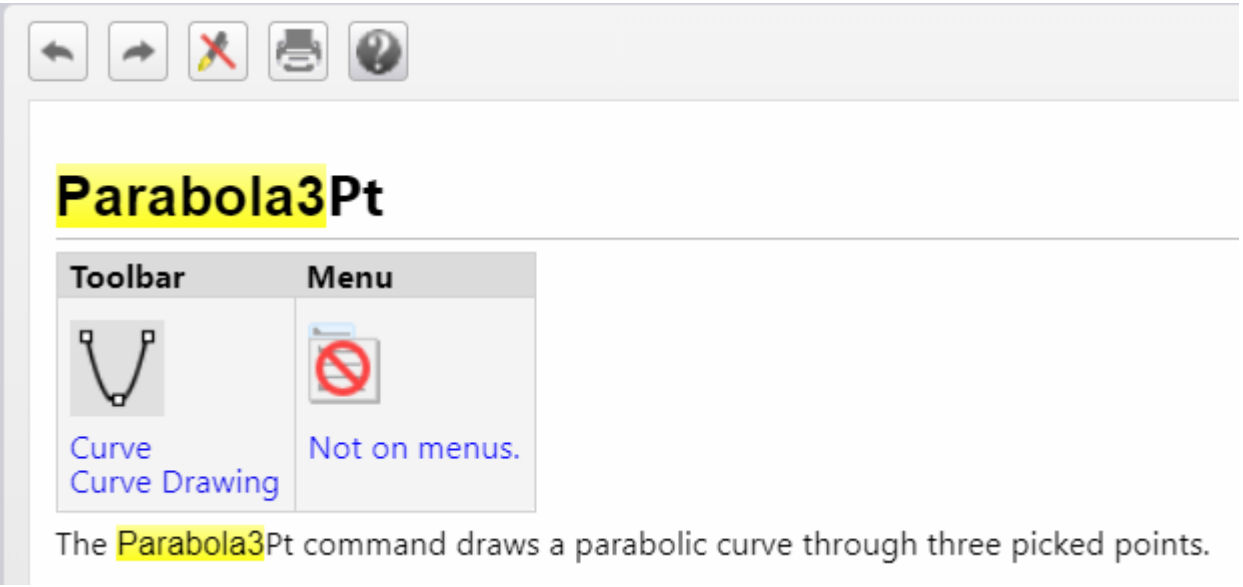
But behold, there is no such thing. Instead, when searching for help, you'll find yourself suggested to write code. Until then, I had never touched scripting in Grasshopper. But the Python code presented by dharman looked pretty straightforward (credit where credit is due).

I took his solution and spent a day to figure out what every line of code meant. Now I know – and want to share it with you.

For an introduction to Rhino3D Grasshopper, read this article. For more learning resources consider this piece of mine.

Rhino3D: Yes / Grasshopper: No

As you can do things in Grasshopper that can't be done in Rhino3D, the same can happen the other way round. As a matter of fact, you actually *can* produce a 3-Point-Parabola in Rhino3D:



But, search in Grasshopper, you won't find such a thing. In my installation there is no Parabola component *at all*, let alone one with a 3-point definition. Maybe there is a solution out there on Food4Rhino but I sure didn't find one.

Solution 1: Use a Rhino3D-Parabola in Grasshopper

So we can easily create a parabola in Rhino3D based on 3 points. A first approach then would be to do this and link it into Grasshopper for further use.

But what if we want to change the parabola's shape? In Rhino3D, we would have to change the position of the points our curve is based on. The curve would change accordingly – but *only* if *Record History* was *on* when we produced the parabola in the first place. Which is not, by default.

Record History is clumsy – especially when compared to Grasshopper's parametric data flow. (GH in 2007 started out to be a refinement of *Record History* anyway.)

Even without this *History* thing, our Rhino3D parabola lacks parametricism – no way to control her 3 points by information-driven inputs.

So *no*, creating the parabola in Rhino3D and linking it to Grasshopper is no option for us.

Solution 2: Create Grasshopper Parabola with Python

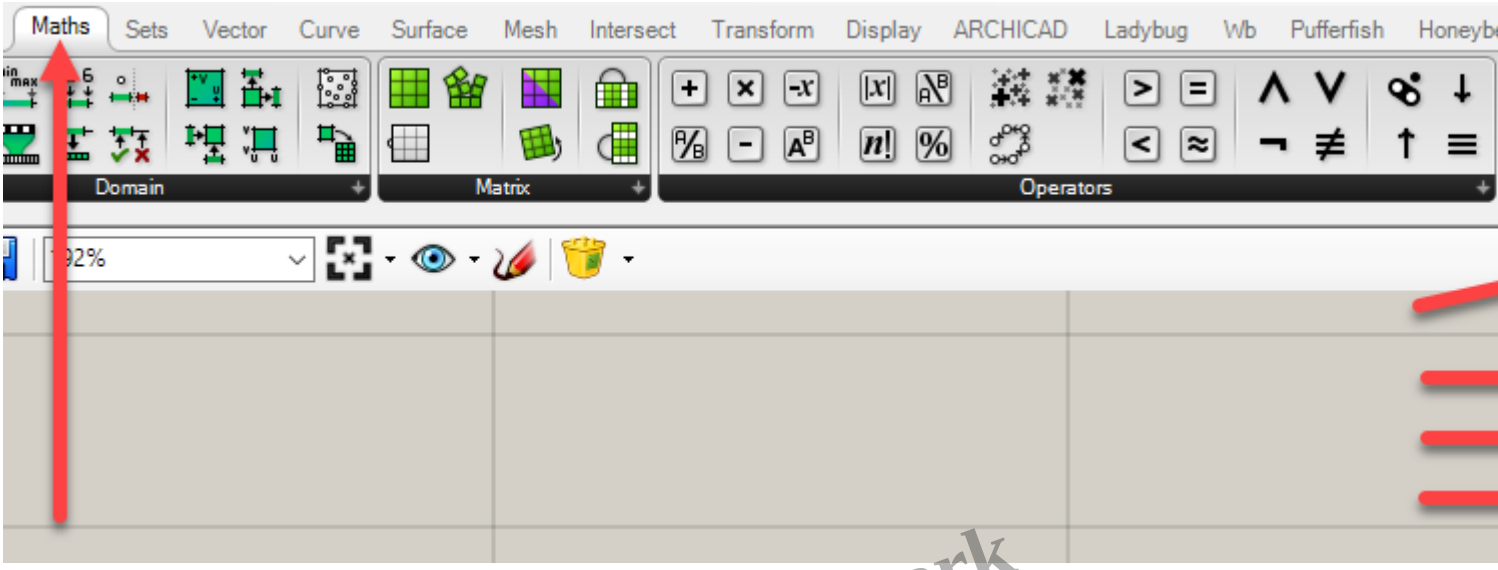
Grasshopper offers a solution that's elegant but asks for writing *code*. This is pretty easy though – at least in this example. Also, it makes sense to get yourself aquainted with it – Grasshopper's functions are *meant* to be enhanced by code.

To produce code in Grasshopper there are tools for

- *C#*

- *Visual Basic (VB)*
- *Python*

You'll find them in *Maths – Scripts:*



I am not a programmer but I know Python offers lots of advantages for amateurs. Most intriguing: You get away with less code. Which is nice because writing code causes bugs – less code, less bugs, accordingly.

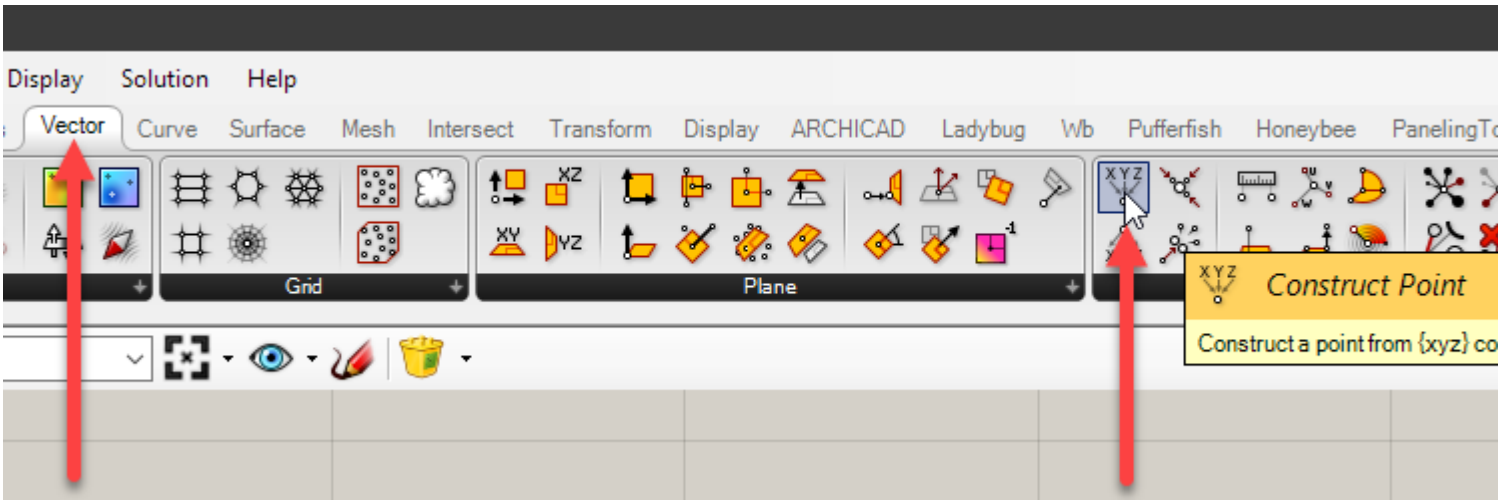
Here we do actually something quite simple: With a *Python* component, we call a *Rhino3D* function inside *Grasshopper*. Which function? The abovementioned *Parabola3Pt* (which we can use to draw a parabola based on 3 points).

New to Grasshopper? I suggest you read this article in the first place.

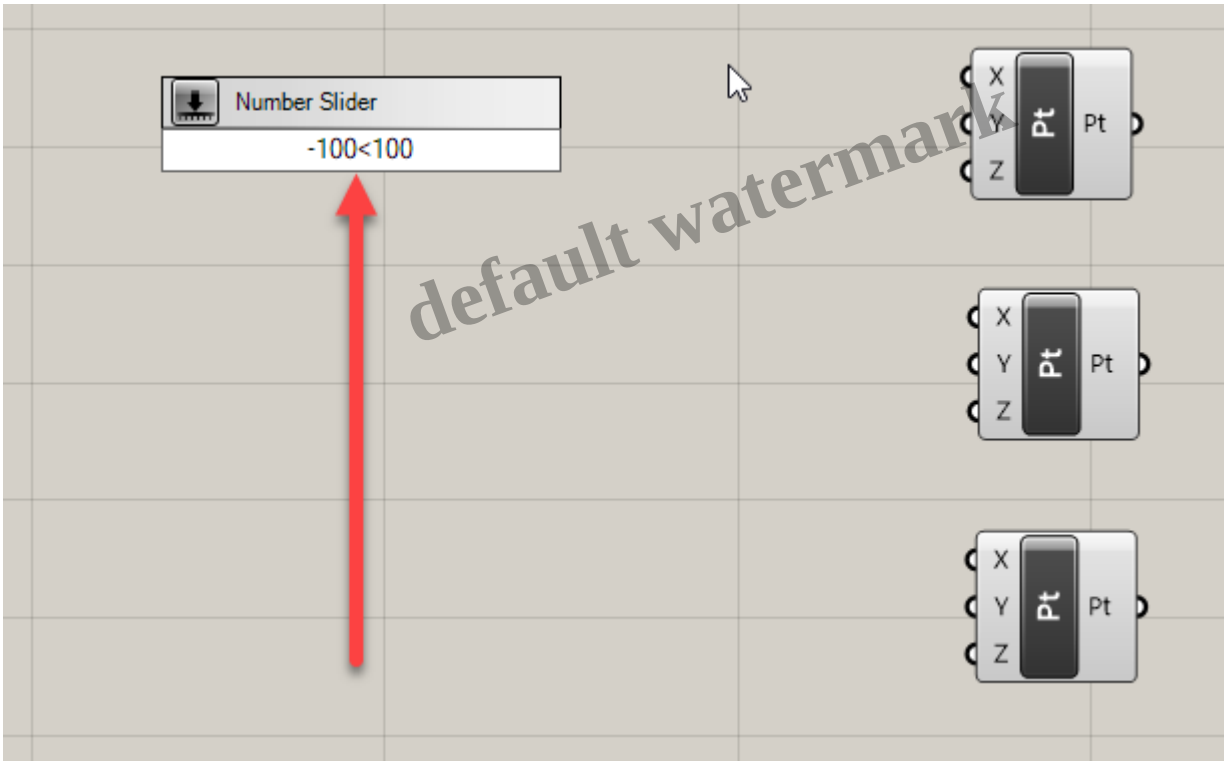
Need more learning resources? Check this out.

3 Points

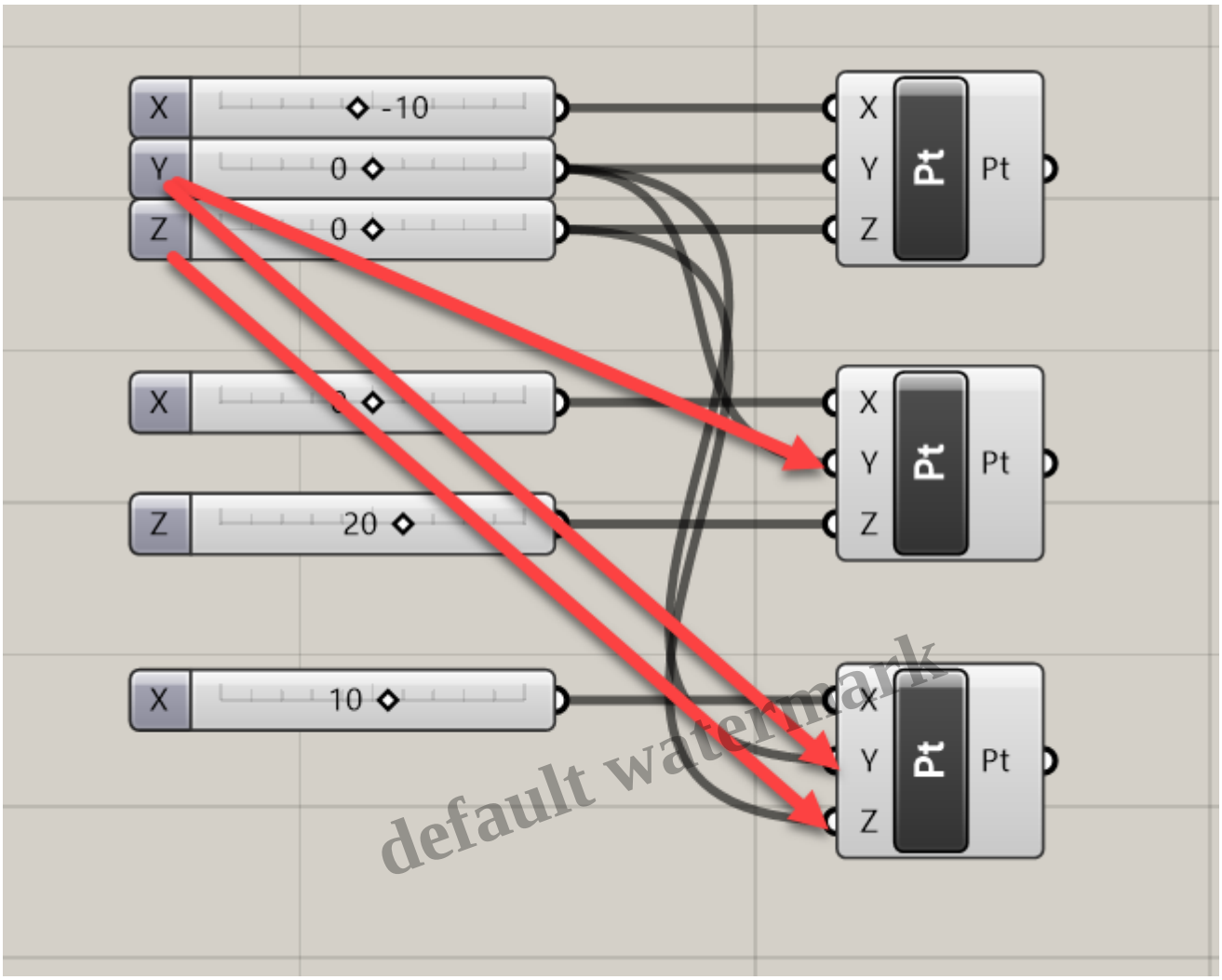
Our parabola will need 3 points, so let's first create these guys. Place 3 *Construct Point* components on your canvas:



Produce *Number Sliders* for coordinates – choose a range you find appropriate:



Connect the *Sliders* to your *Point* components. Be aware inputs may share the same value – use *Sliders* for more than one input where it makes sense:



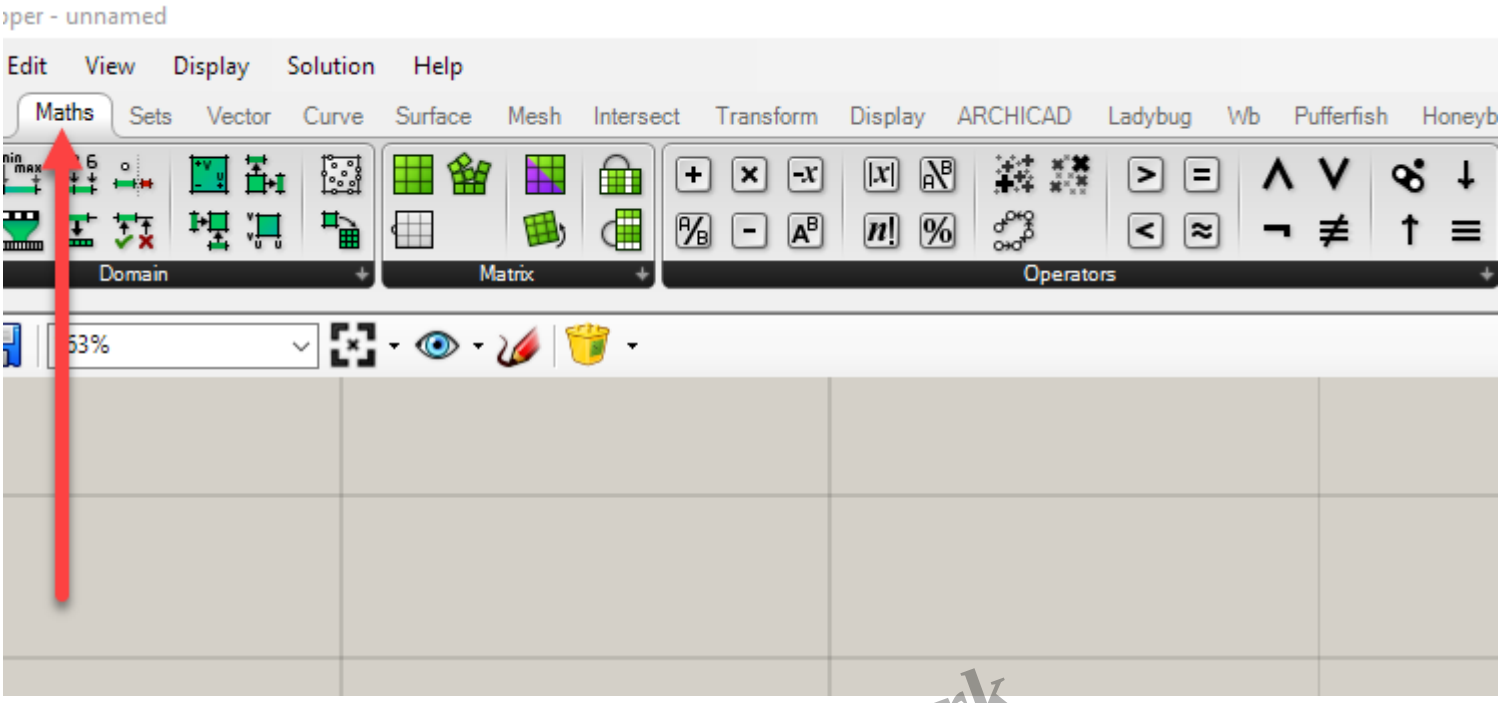
Depending on your *Slider* settings, the point group may look like this:

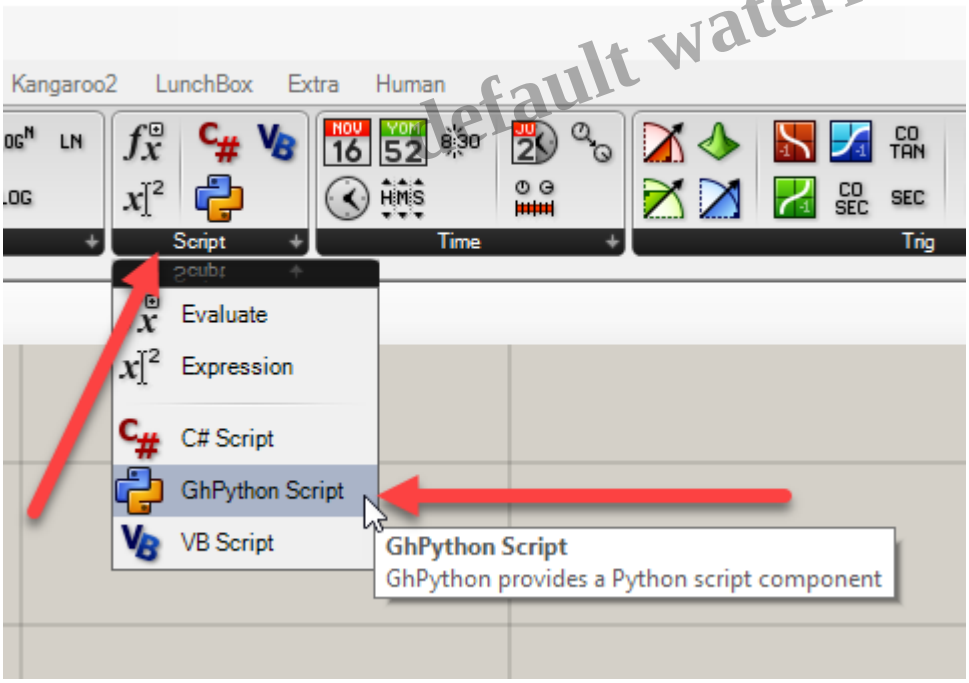
New to Grasshopper? I suggest you read this article in the first place.

Need more learning resources? Check this out.

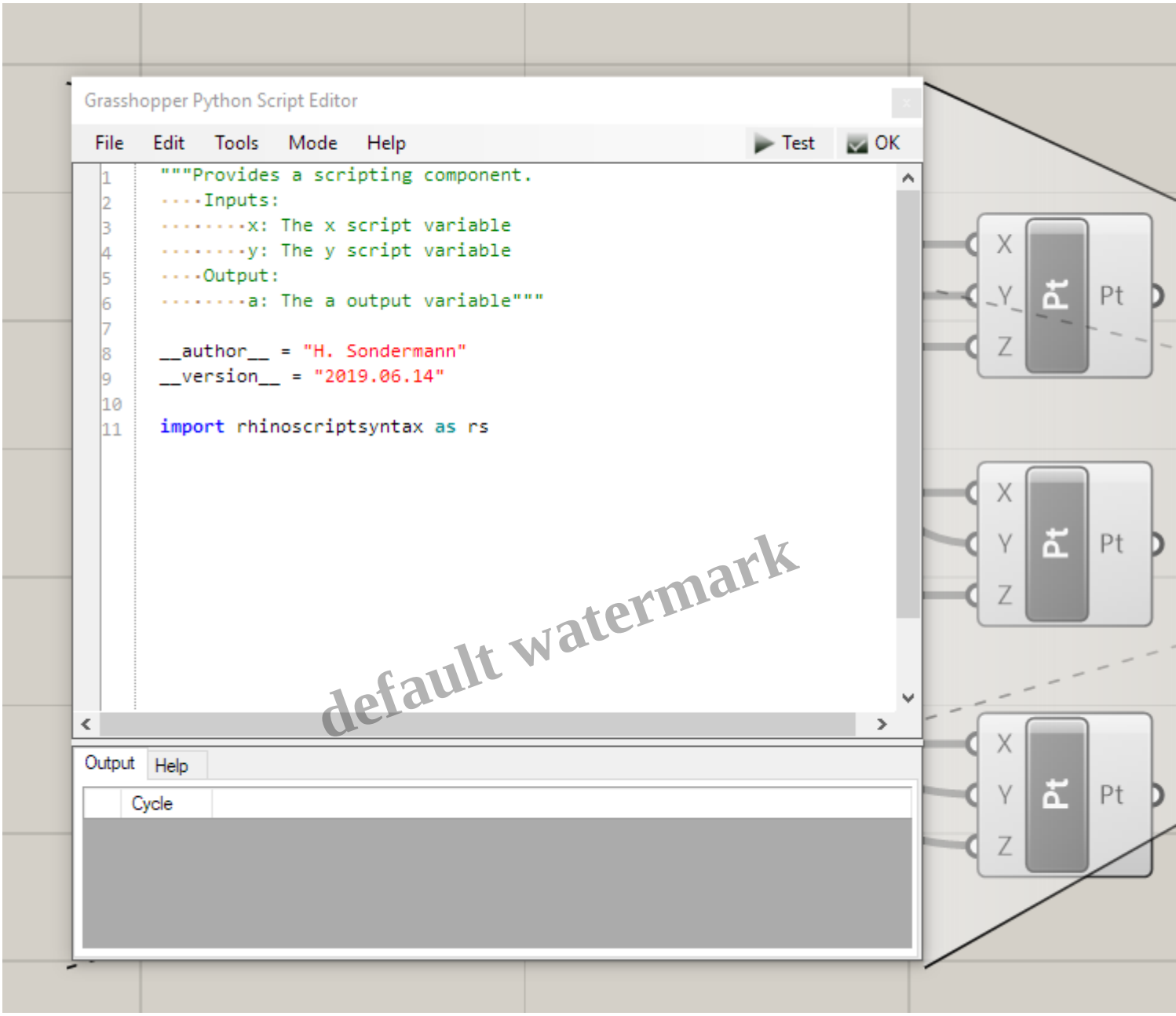
Python component: Code

Let's tackle the *code* part. Grab a *GHPython Script* component:





Double-click the component, this window opens:

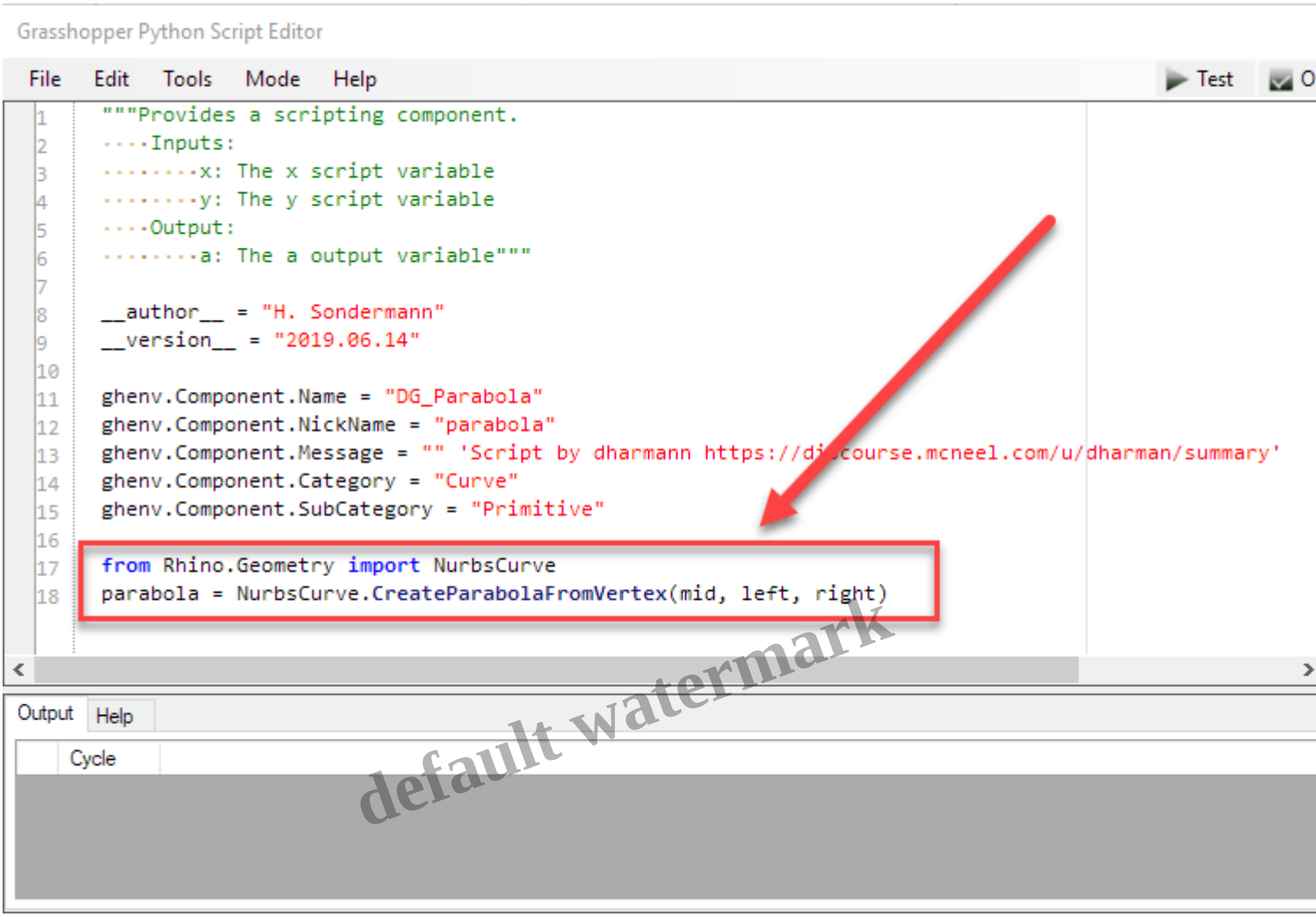


Here we are – this is actually what you see when you open a fresh *Python* component. (You won't see *my* name, of course – this one comes from your Grasshopper *Author* preferences. And also the *version* date will differ.)

Now delete the last line beginning with *"import …".* Instead, paste this:

from Rhino.Geometry import NurbsCurve

parabola = NurbsCurve.CreateParabolaFromVertex(mid, left, right)



This is the *code*. The rest (everything above those 2 lines) is useful, but not necessary to create the parabola.

The green text on top is set between triple brackets. It is a comment, meaning it carries info but doesn't do anything. Here you can explain what the component will do. (Yes, you will produce a *real* component that *does* something. And it is good practice to inform people about it.)

By the way: Grasshopper will display the first line when you mouse over your component.

So it makes sense to write something meaningful here – this, for example:

Provides a Parabola 3Pt component

Inputs:

left: The "left" base point

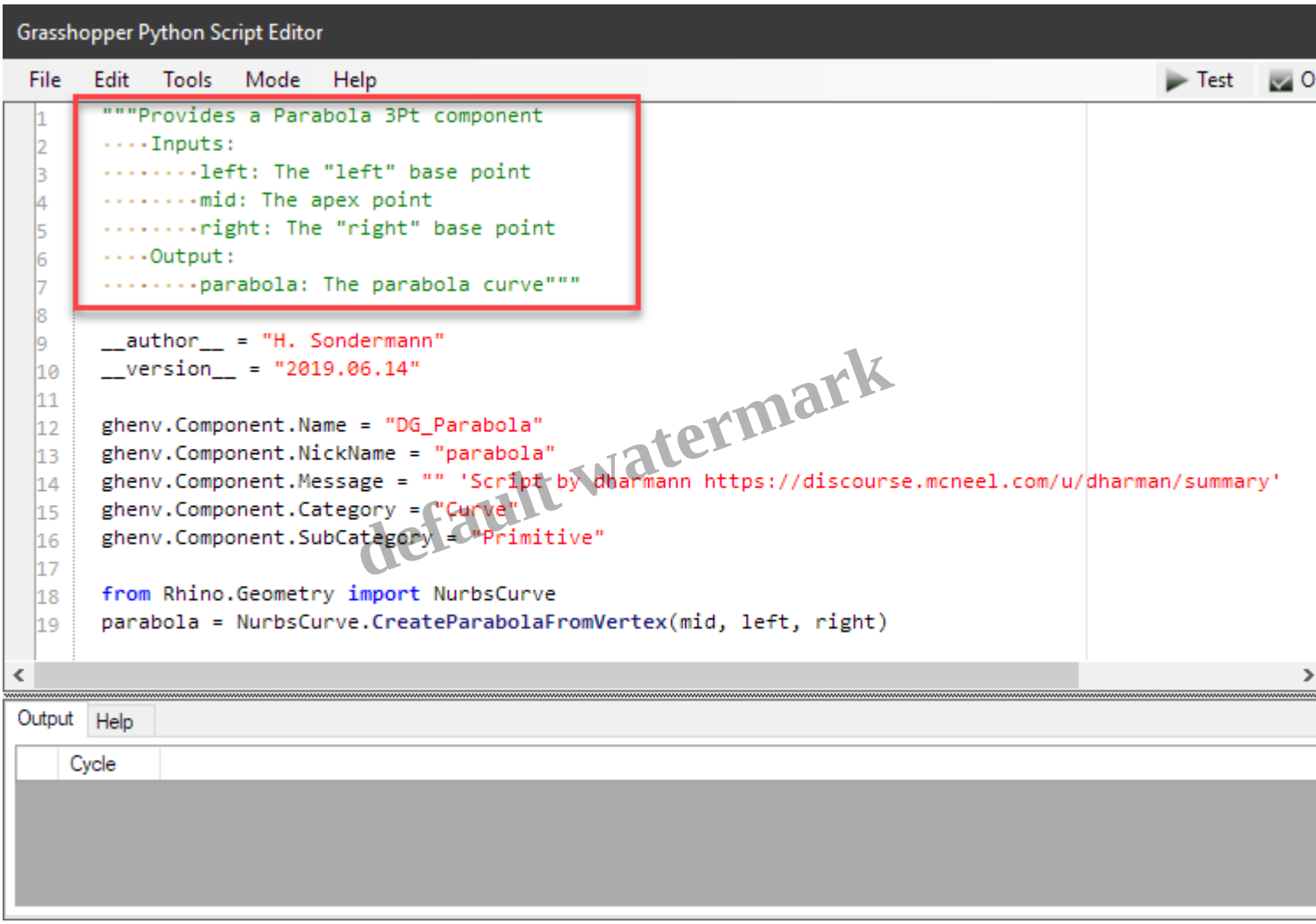
mid: The apex point

right: The “right” base point

Output:

parabola: The parabola curve

Just copy the above lines and paste them between the brackets:



In the middle you see 5 more lines that do nothing for creating our parabola. But they help naming and saving the component properly.

Again, feel free to copy these lines and paste them:

ghenv.Component.Name = “DG_Parabola”

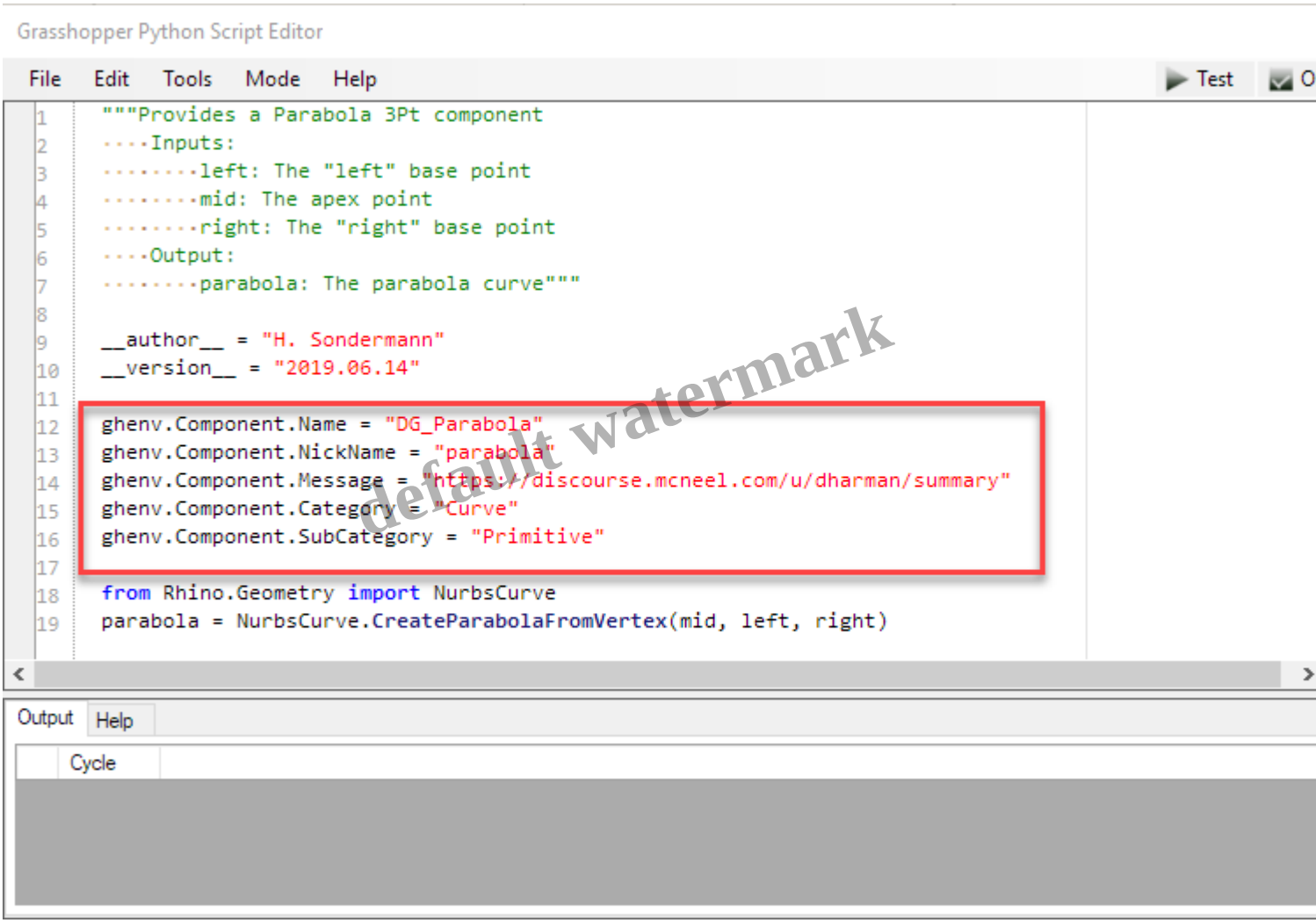
ghenv.Component.NickName = “parabola”

ghenv.Component.Message = “https://discourse.mcneel.com/u/dharman/summary”

ghenv.Component.Category = “Curve”

ghenv.Component.SubCategory = “Primitive”

The first line creates the component’s name. Second line creates the name that’s written on it (when *Draw Icons* in *Display* is unchecked). The third line produces a tiny label below the component (here: the URL where I found the script, see above). The fourth and fifth line help sorting the component into the right menu, should you save it for eternal use:



New to Grasshopper? I suggest you read this article in the first place.

Need more learning resources? Check this out.

Python component: Inputs + Outputs

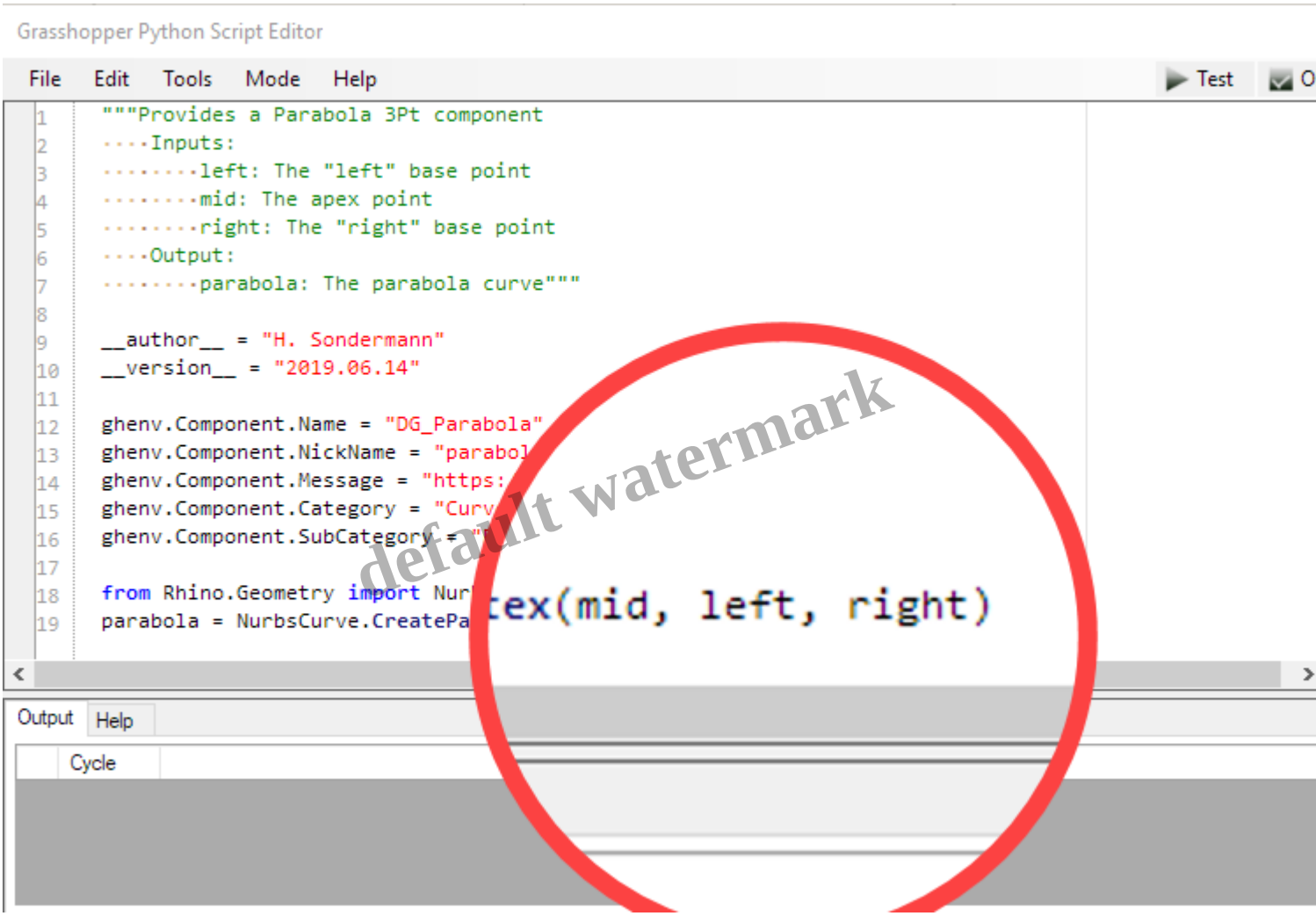
Code’s finished. By the way we didn’t talk about the actual *meaning* of it. Don’t worry, we’ll come to that. I just want to hurry so you see your parabola eventually.

Now, still in the scripting window, press the *Test* button top right – you’ll get an error message. This is *normal*

. Scripting may be simpler than thought, but it is still a major exercise in mindfulness.

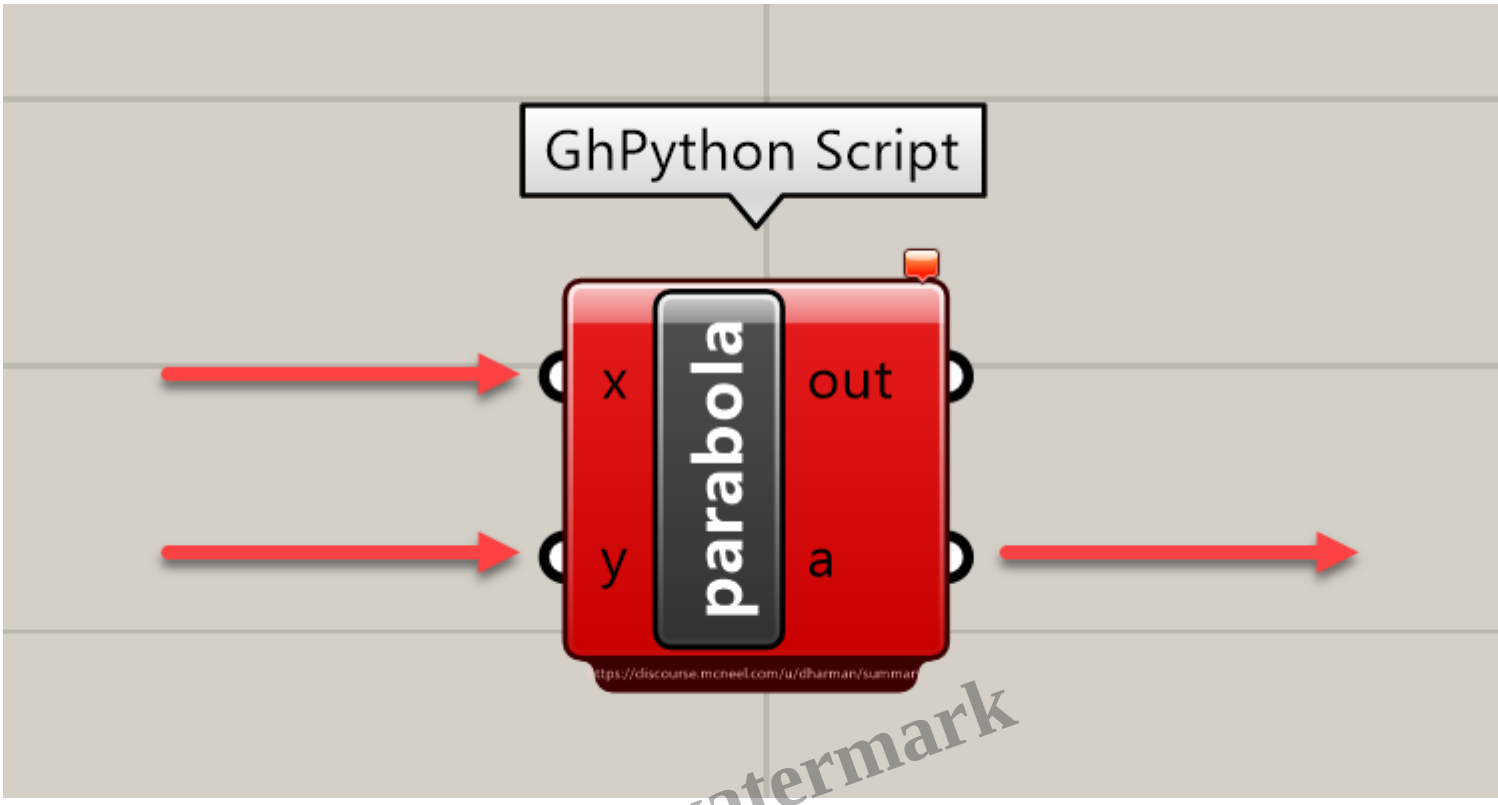
I assume you already identified the last line (starting with *parabola* = …) as the actual function that creates the parabola. And you're right.

Now if you got this, you also understand that *parabola* is the component's output. And: *mid, left, right* are the according inputs:



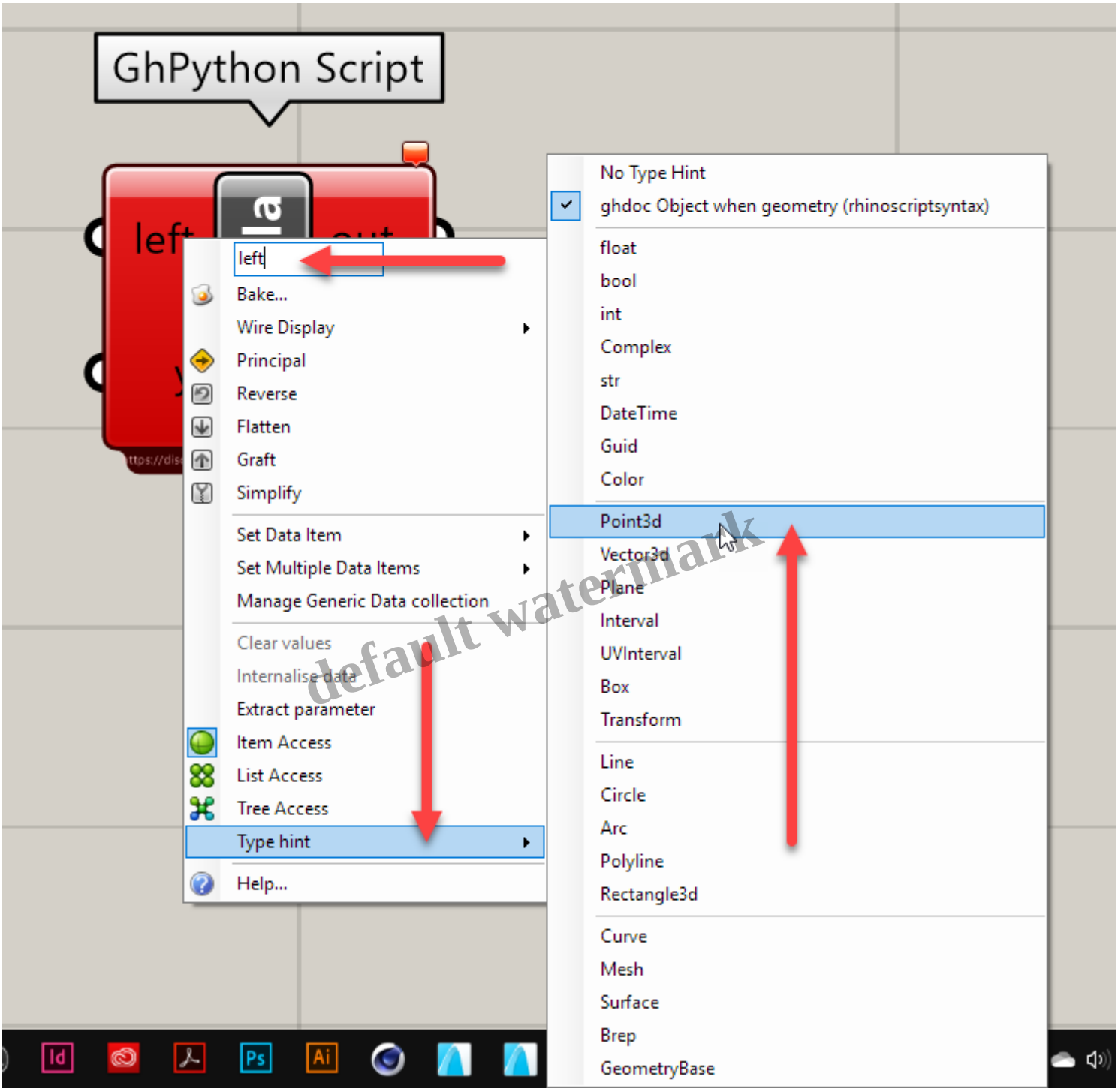
If you look at the comment above you see this confirmed.

Now if you click OK and take a look at your component you won't find those words on it:



Inputs are named *x* and *y*, the output's name is *a*. (*out* is not the component's output, we'll get rid of it later.) And because the component's inputs and outputs don't match the variables in the code, the component appears *red* – not a good sign at all.

To fix this, right-click on *x* and change some settings:

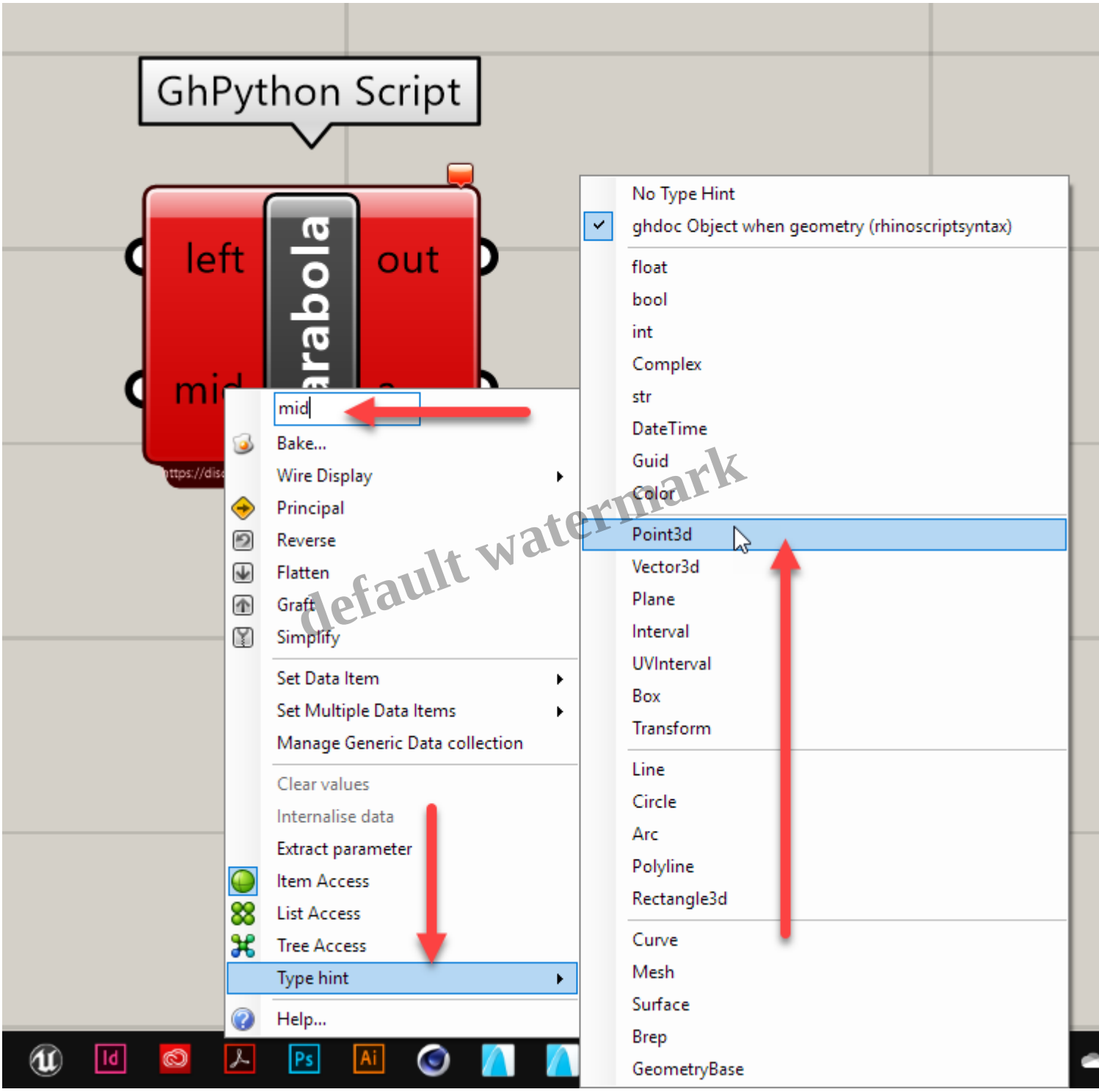


First, the name – take *left*, for example. You could also use *mid* or *right* – what is important though is that each name is exactly written as it is in the code. Python is case-sensitive!

Then, change *Type hint* to *Point3d*. Python needs this setting, otherwise it can't utilize the point input.

Done.

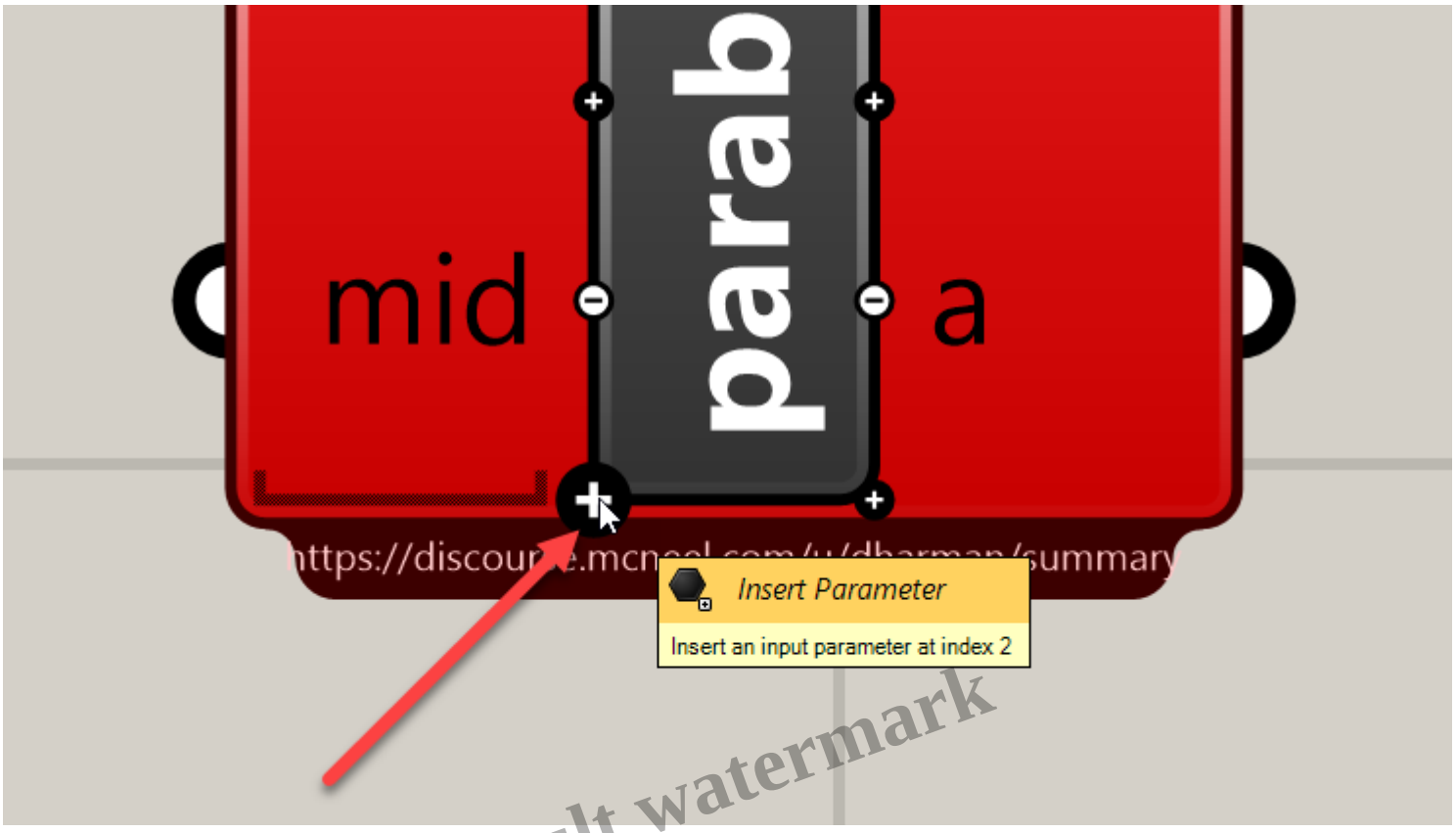
Repeat this for *y:*



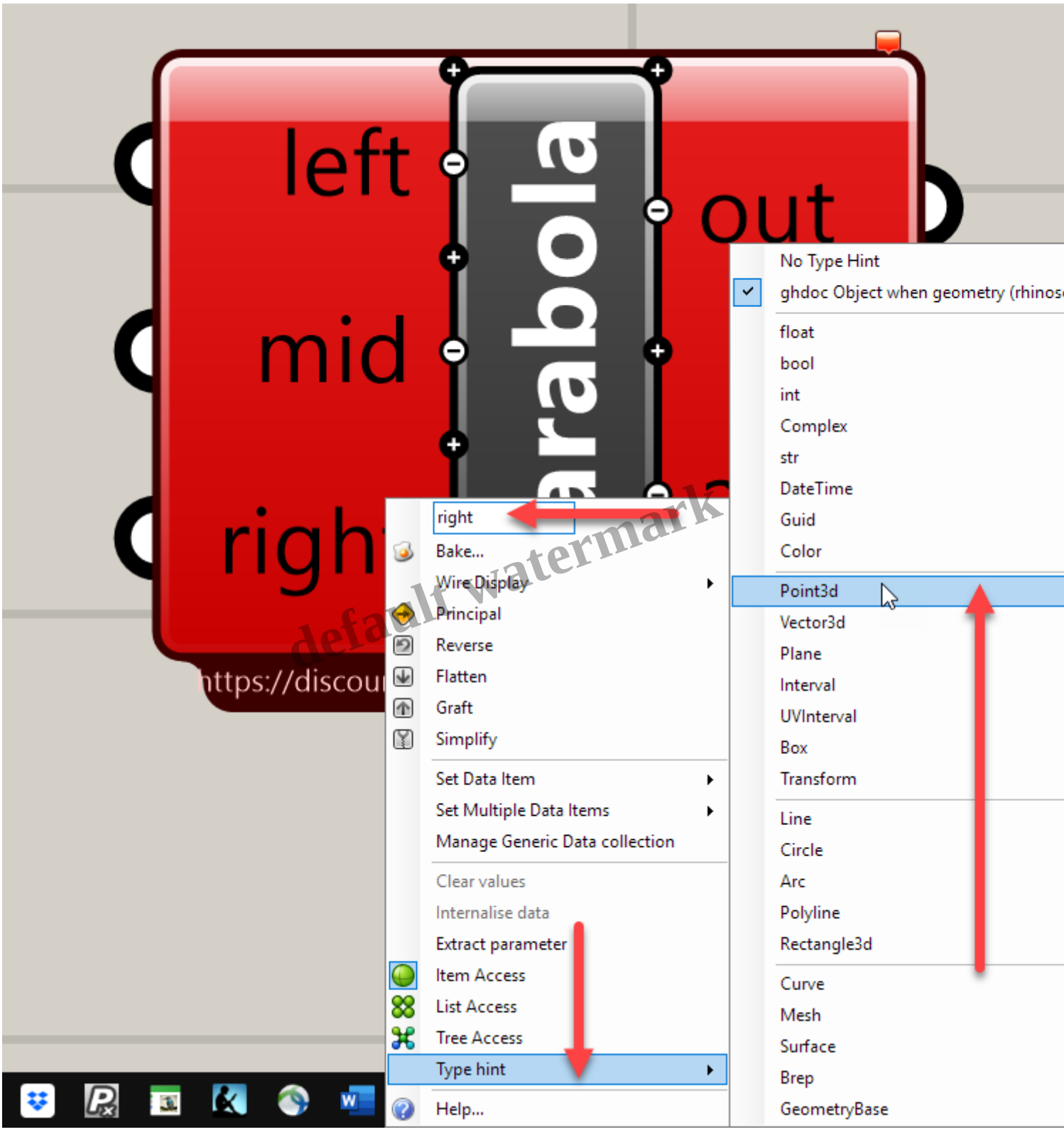
Name could be *mid*, and don't forget *Type hint Point3d*.

Done.

Now for the third input, zoom in on the component, until you see a little plus sign:



Click on it to produce a third input, then proceed the way you did with *x* and *y:*



As you might have guessed, this one will be *right*. And: *Type hint Point3d*, again.

Done.

Now rename output *a* to *parabola*. (Again, respect Python's case sensitivity.)

No *Type hint* issues here, so you are done here, too.

Finally, you may remove outpout *out.* (Zoom in and click the *minus* sign.) It is not needed for regular Grasshopper work and will only mislead you eventually.

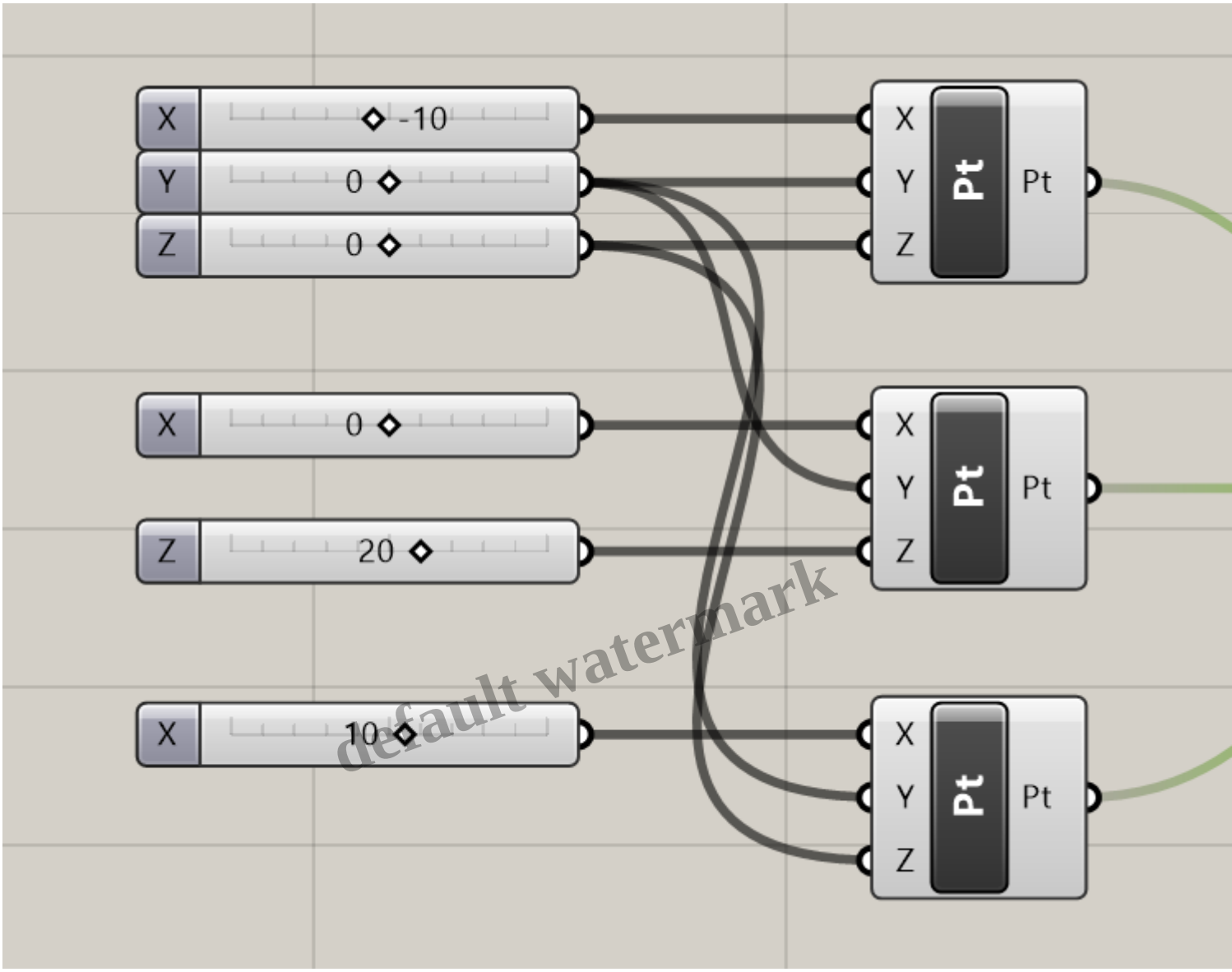
New to Grasshopper? I suggest you read this article in the first place.

Need more learning resources? Check this out.

Component creates Parabola

Everything should be fine now. *Test* your code again, there shouldn't be any error message anymore. If there is, check every detail, take your time. Debugging is an art in itself, it doesn't come easy.

Our *Construct Point* components have been waiting patiently – finally we can connect them to our selfmade *Parabola* component:



And here we are:

Isn’t this great? Now move your Number Sliders to and fro and watch your Parabola change its shape.

New to Grasshopper? I suggest you read this article in the first place.

Need more learning resources? Check this out.

Python Code Whisper

You see what can be achieved with some lines of code in Grasshopper. Now lets have a word about the meaning of this code.

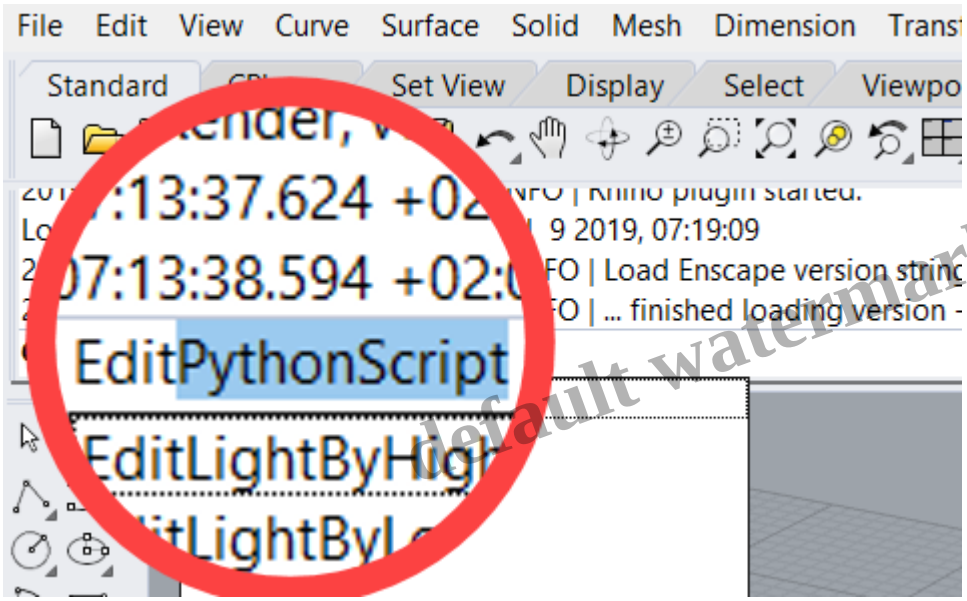
(You feel inspired to read more about this subject? Download and work yourself through the Rhino Python Primer

. Too long? Check this out.)

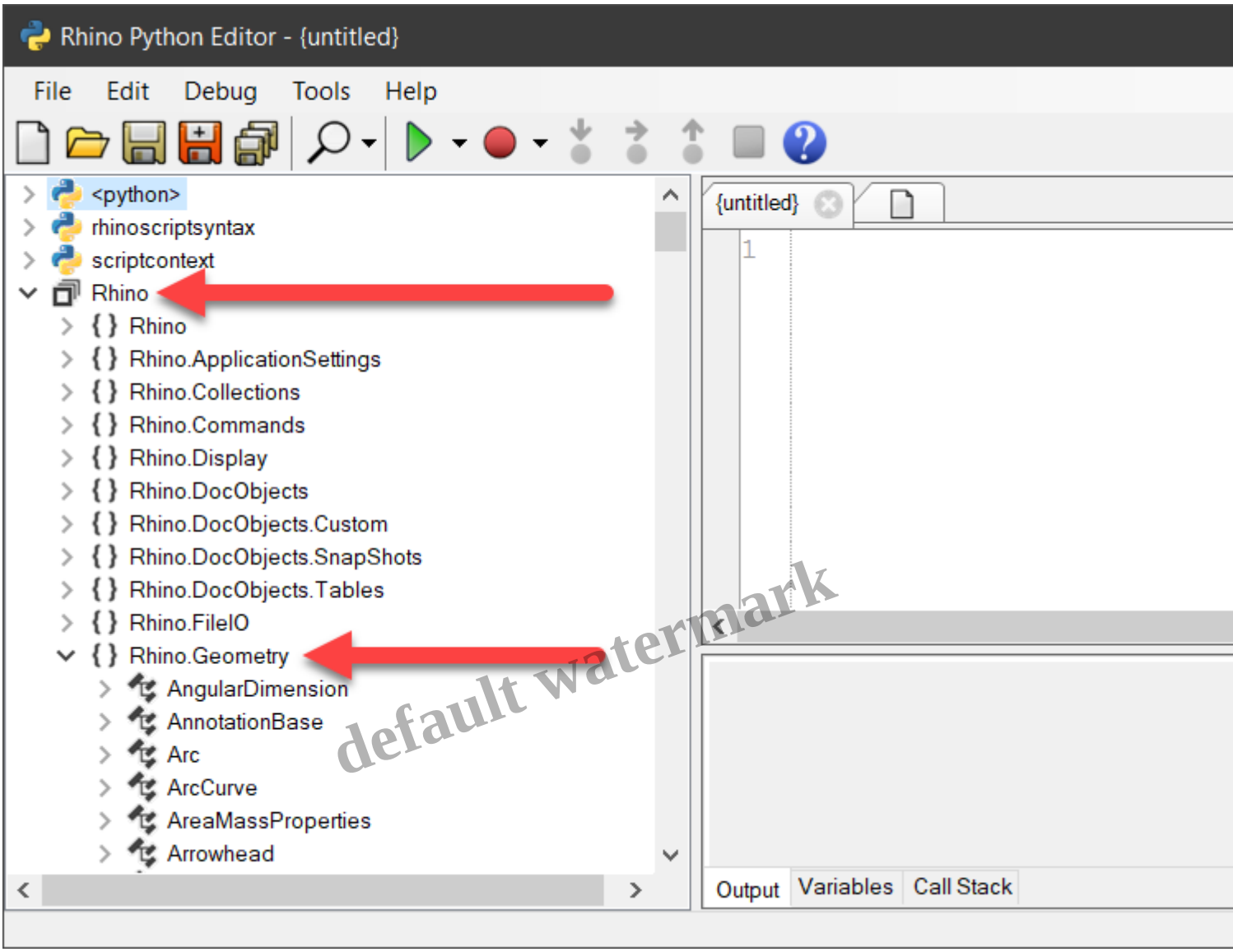
As said before: The code that actually *creates* the parabola is made up of *2* lines only:

- *from Rhino.Geometry import NurbsCurve*
- *parabola = NurbsCurve.CreateParabolaFromVertex(mid, left, right)*

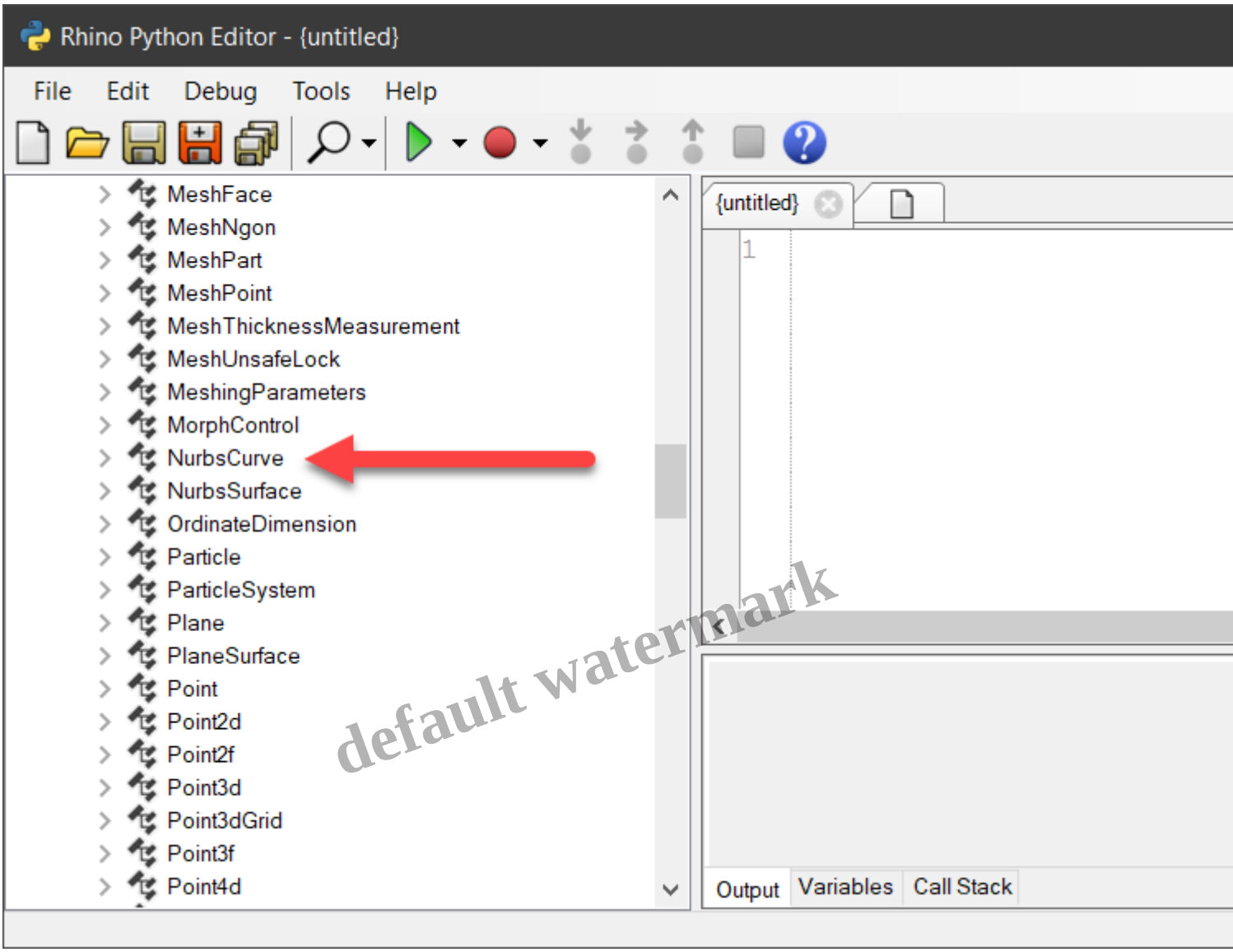
Let's look at the first line: It's a command to import a set of methods. Where from? Enter *EditPythonScript* in Rhino3D's command prompt:



A window opens – it's a script editor again, this time in Rhino3D. On the left is a navi with some lists. One is named *Rhino:*



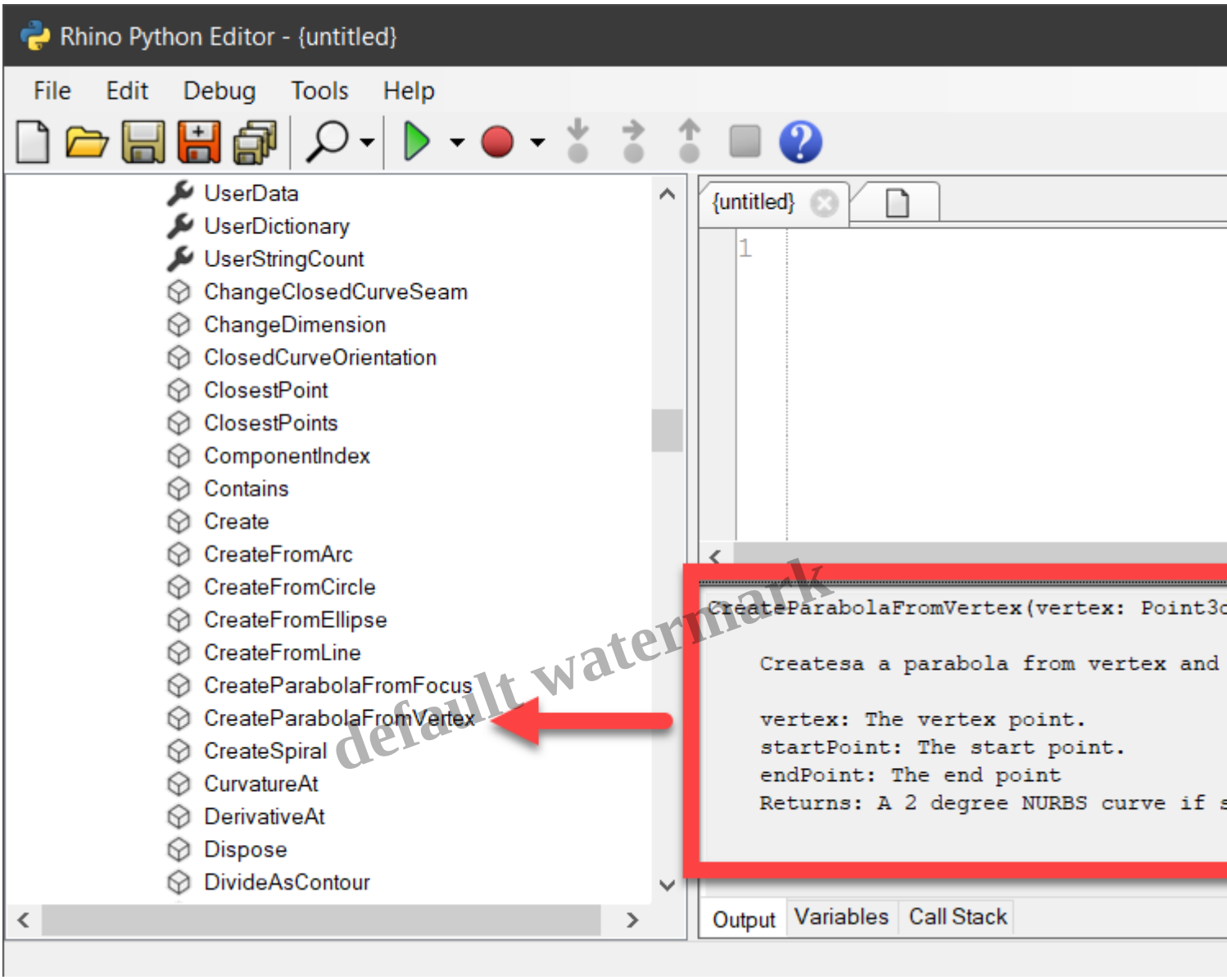
There's also a submenu called *Rhino.Geometry*. In our code, we tell Grasshopper to import *from Rhino.Geometry,* so this is it. Further down you find our *NurbsCurve:*



Again this is not a single object (or rather *method)* but a subfolder.

Here we are, this is what our first line of code does: It tells Grasshopper to use something out of this *NurbsCurve* folder which again is an item inside *Rhino.Geometry*.

Next step: We pick one of the methods stored in this *NurbsCurve* folder:



Again, you see this is exactly what our 2nd code line contains:
parabola = NurbsCurve.CreateParabolaFromVertex(mid, left, right).

Meaning: Produce an output called *parabola* with a *Rhino.Geometry* method called *NurbsCurve.CreateParabolaFromVertex.*

The script editor even shows an instruction how the method is to be used: It takes three *Point3d* variables (remember: *Type hint!*). The first being the mid* point (vertex), the other two start and end points.

Important: *vertex, startPoint* and *endPoint* are placeholder names. In our example we called them *mid, left* and *right*. It was crucial though that these names matched the components input names.

That's it.

(**Mid* isn't meant literally. It's just some point between start and end.)

Bottom Line?

If you don't find a function in Grasshopper that you know exists in Rhino3D – use Python Code.

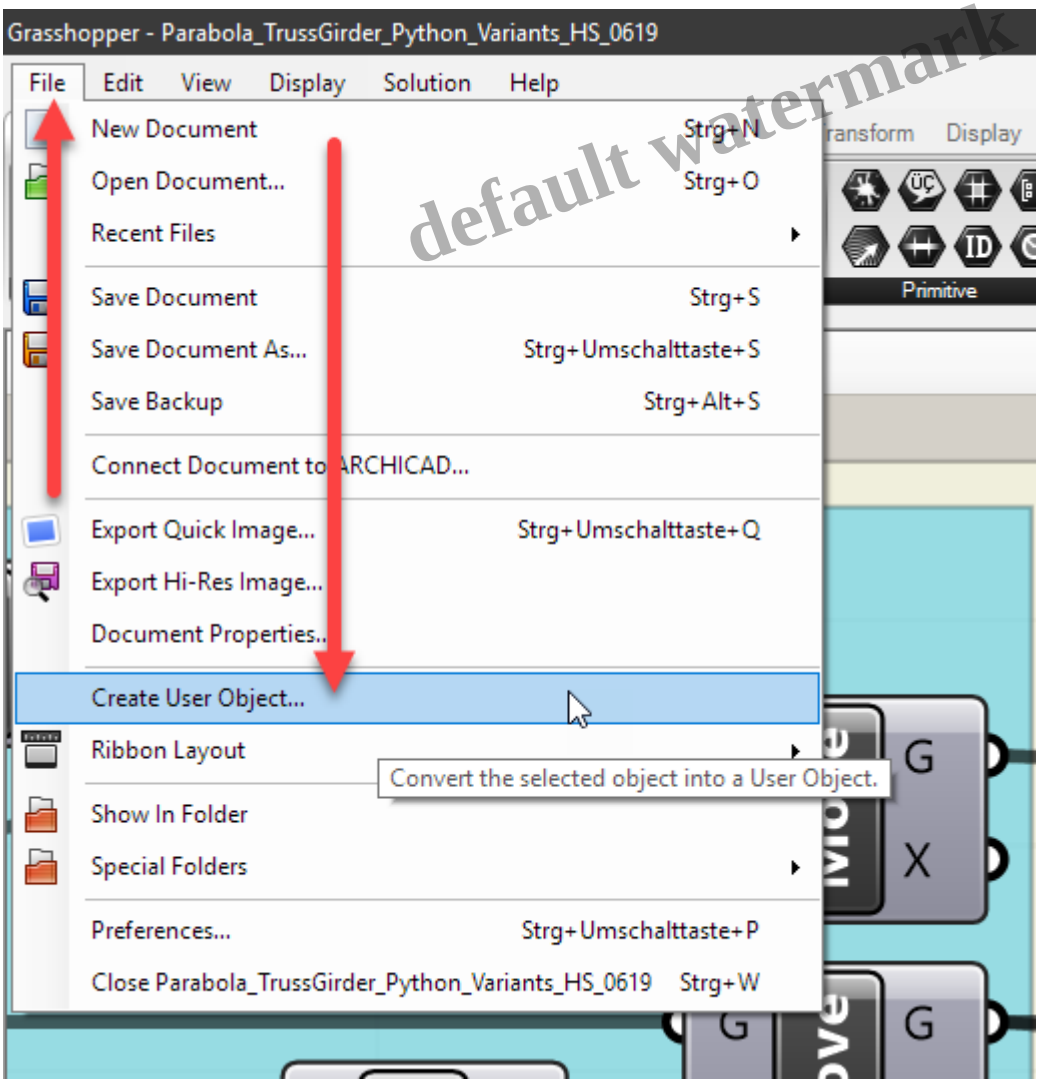
New to Grasshopper? I suggest you read this article in the first place.

Need more learning resources? Check this out.

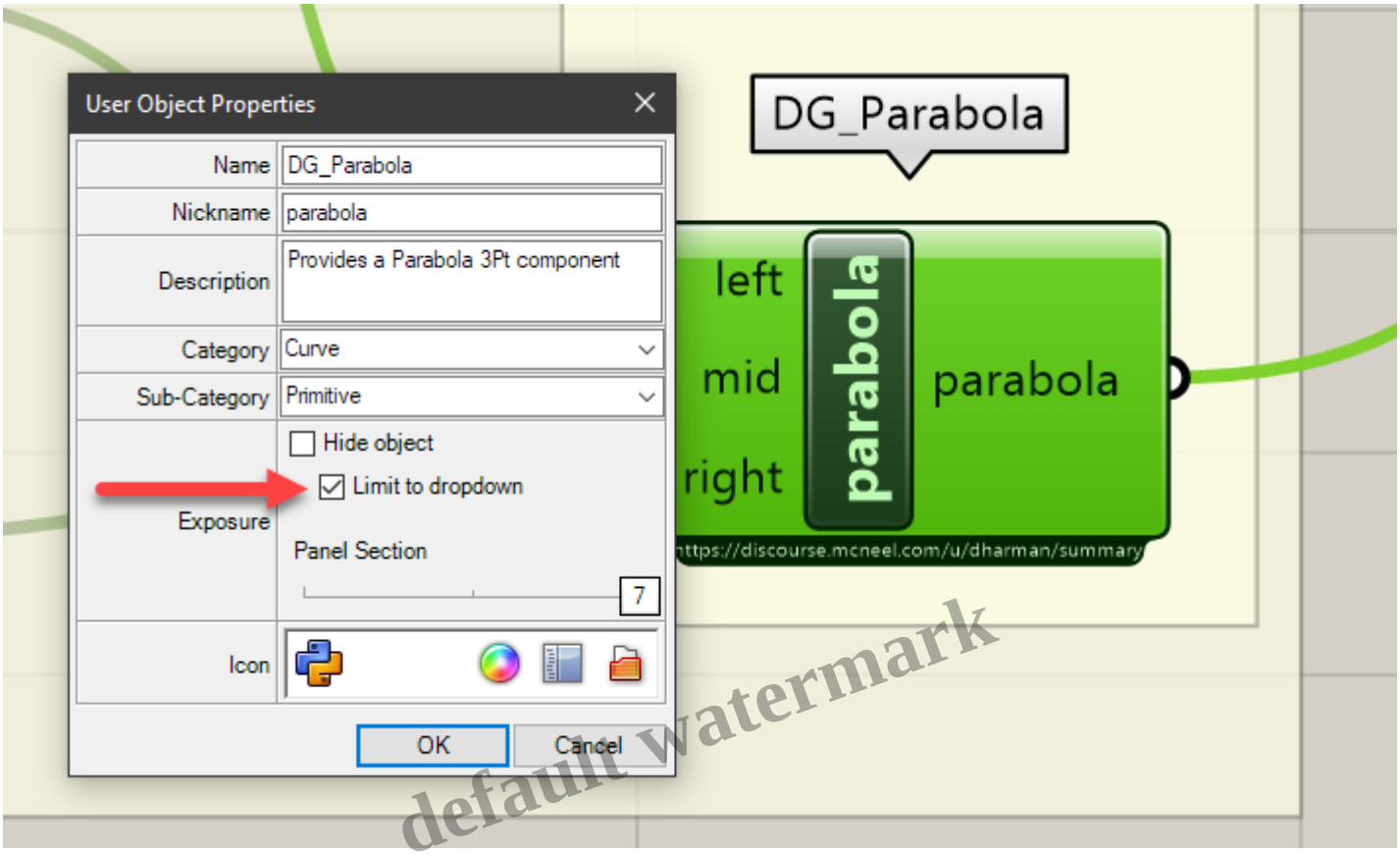
Parabola3Pt Component Forever

Now as I promised, I show you how to save your Parabola component. In a way that allows you to use it like every other component in your Grasshopper menus. You can even share it with others and thus contribute to the community of Rhino3D/Grasshopper users worldwide.

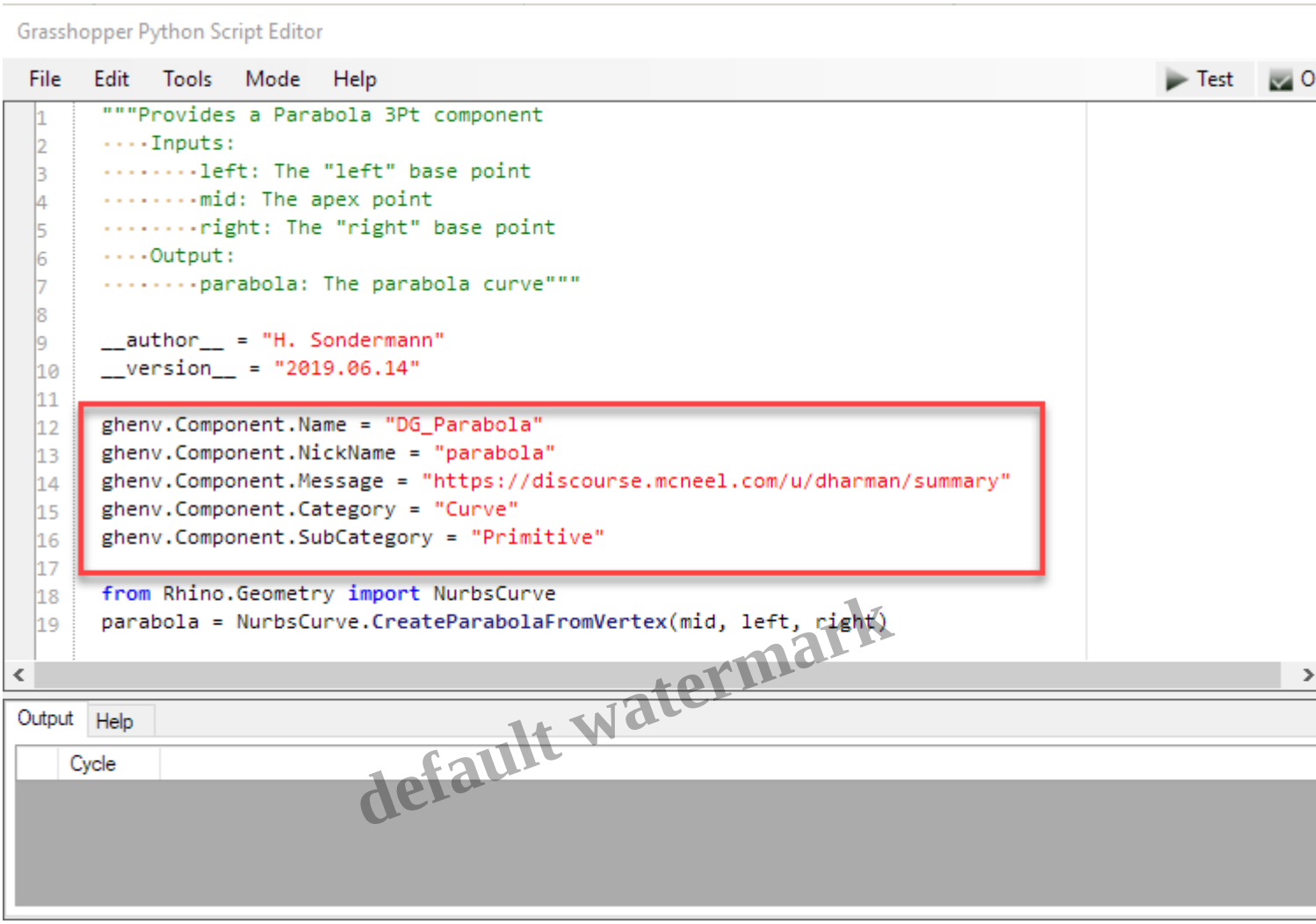
Actually it's quite simple. Just select your Python component and hit *File – Create User Object …*



A window opens:



Now you see why you filled in the *ghenv* lines in the Script window – remember?

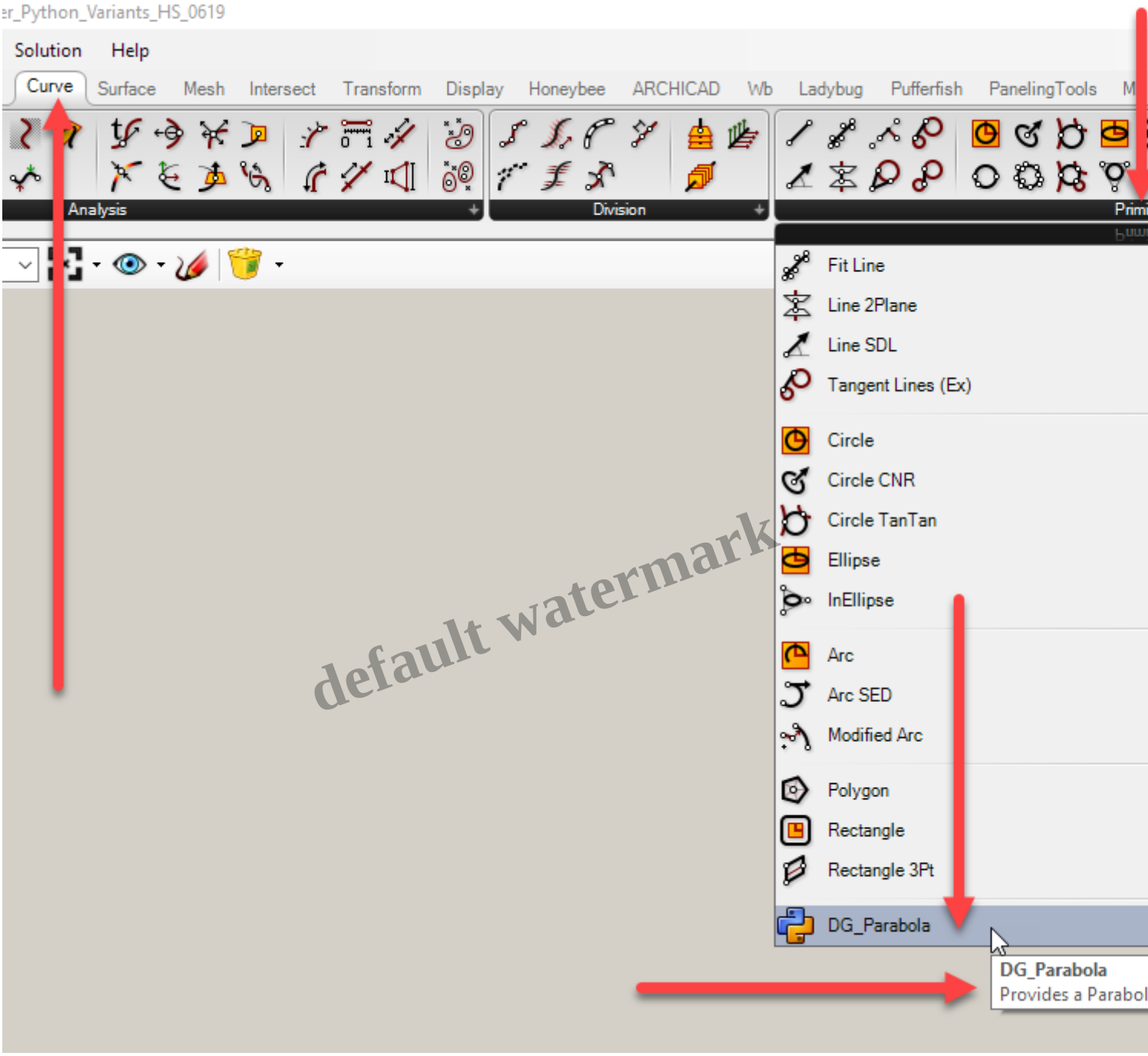


Because their info is filled into the component save settings *(Name, NickName, Category, SubCategory)*. Only the *desription (Provides a Parabola 3Pt component)* is taken from elsewhere: the first comment line right at the top.

Next, you can decide if the component will be hidden. (So you can only reach it via double-click-and-write-its-name-on-the -canvas). And if it's not hidden, whether it will be shown only in the dropdown menu.

Last not least, you are invited to choose an icon for the components menu appearance. Check it out!

After saving, you will find your Parabola component at the respective location:



And as for the file: Choose *File – Special Folders – User Object Folder* (in Grasshopper!). Your file system shows up, and your component should be listed here. (File extension: *.ghuser).* This is a file you can share – people may copy it into the same folder on their HD, or even drag it onto their Grashopper canvas to install it on their system as well.

New to Grasshopper? I suggest you read this article in the first place.

Need more learning resources? Check this out.

Roundup

Grasshopper functionality can be extended with code. This creates endless potential and may well ask for lots of programming skills.

But for you and me, first and easy steps may be just to recreate Rhino3D methods inside Grasshopper that don't come with a component.

Again, for a deeper understanding read the Rhino Python Primer. For an intro into Grasshopper, read my introduction.

Category

1. Rhino/Grasshoppper

Tags

1. 2D Geometry
2. Parametric Modeling

Date Created
July 2019
Author
hsondermanncom