Python Basics

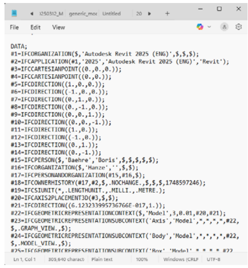
DIM Python Basics 01 (Introductie)

03.11.2025

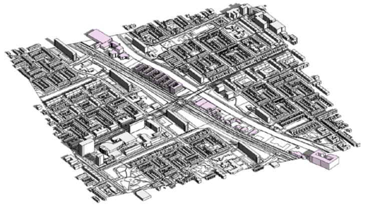
Hanze

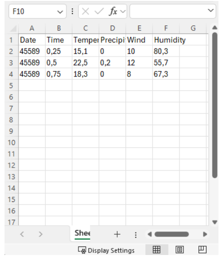
0.1 Why Coding in BE?

1. **Everything is data now** (CSV, IFC, 3D Bag, GIS, sensor data, EPW)
2. Instead of organizing layers and steps the new technology asks **knowledge about iterations and objects**
3. **Software never fits** your exact workflow (Revit, Rhino, Civil 3D, QGIS, ACC, Navisworks)
4. Parametric / generative **design based on objects** is becoming standard (Revit/Dynamo, Rhino/Grasshopper)
5. Interoperability needs automated adjustments
6. To check sustainability & performance **we need to simulate and analyse** by code
7. Construction is **increasingly automating**
8. It makes you **more valuable** in teams
9. The industry is moving towards "information management"
10. We need to do more in the same or better quality with less hands

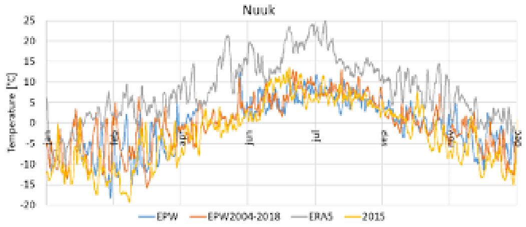


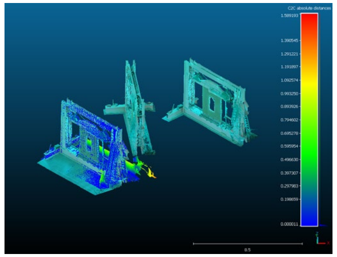
```
DATA;
#1=IFCORGANIZATION($,'Autodesk Revit 2025 (ENG)',$,$,$);
#2=IFCAPPLICATION(#1,'2025','Autodesk Revit 2025 (ENG)','Revit');
#3=IFCCARTESIANPOINT((0.,0.,0.));
#4=IFCCARTESIANPOINT((0.,0.));
#5=IFCDIRECTION((1.,0.,0.));
#6=IFCDIRECTION((-1.,0.,0.));
#7=IFCDIRECTION((0.,1.,0.));
#8=IFCDIRECTION((0.,-1.,0.));
#9=IFCDIRECTION((0.,0.,1.));
#10=IFCDIRECTION((0.,0.,-1.));
#11=IFCDIRECTION((1.,0.));
#12=IFCDIRECTION((-1.,0.));
#13=IFCDIRECTION((0.,1.));
#14=IFCDIRECTION((0.,-1.));
#15=IFCPERSON($,'Baehre','Boris',$,$,$,$,$);
#16=IFCORGANIZATION($,'Hanze','',$,$);
#17=IFCPERSONANDORGANIZATION(#15,#16,$);
#18=IFCOWNERHISTORY(#17,#2,$,.NOCHANGE.,$,$,$,1748597246);
#19=IFCSIUNIT(*,.LENGTHUNIT.,.MILLI.,.METRE.);
#20=IFCAXIS2PLACEMENT3D(#3,$,$);
#21=IFCDIRECTION((6.123233995736766E-017,1.));
#22=IFCGEOMETRICREPRESENTATIONCONTEXT($,'Model',3,0.01,#20,#21);
#23=IFCGEOMETRICREPRESENTATIONSUBCONTEXT('Axis','Model',*,*,*,*,#22,
$,.GRAPH_VIEW.,$);
#24=IFCGEOMETRICREPRESENTATIONSUBCONTEXT('Body','Model',*,*,*,*,#22,
$,.MODEL_VIEW.,$);
#25=IFCGEOMETRICREPRESENTATIONSUBCONTEXT('Box','Model',*,*,*,*,#22
```



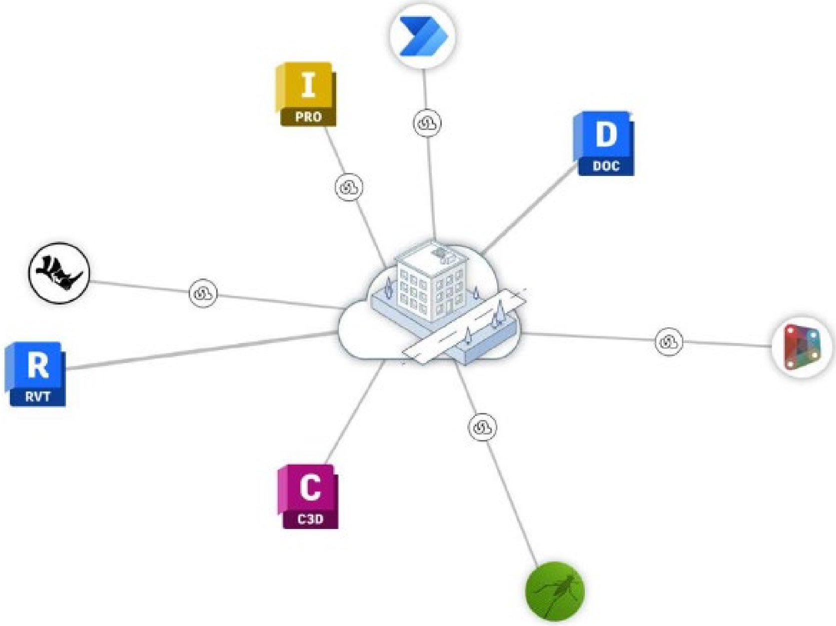


	A	B	C	D	E	F	G
1	Date	Time	Temper	Precipi	Wind	Humidity	
2	45589	0,25	15,1	0	10	80,3	
3	45589	0,5	22,5	0,2	12	55,7	
4	45589	0,75	18,3	0	8	67,3	
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							





0.2 Reasons in short



1. You don’t learn to code to become a programmer –

   but you need to understand the possibilities.

2. You learn to code

   because BE structures are becoming **information systems.**

3. If you can’t control the information,

   you can’t control **the process to build** something.

0.3 Levels

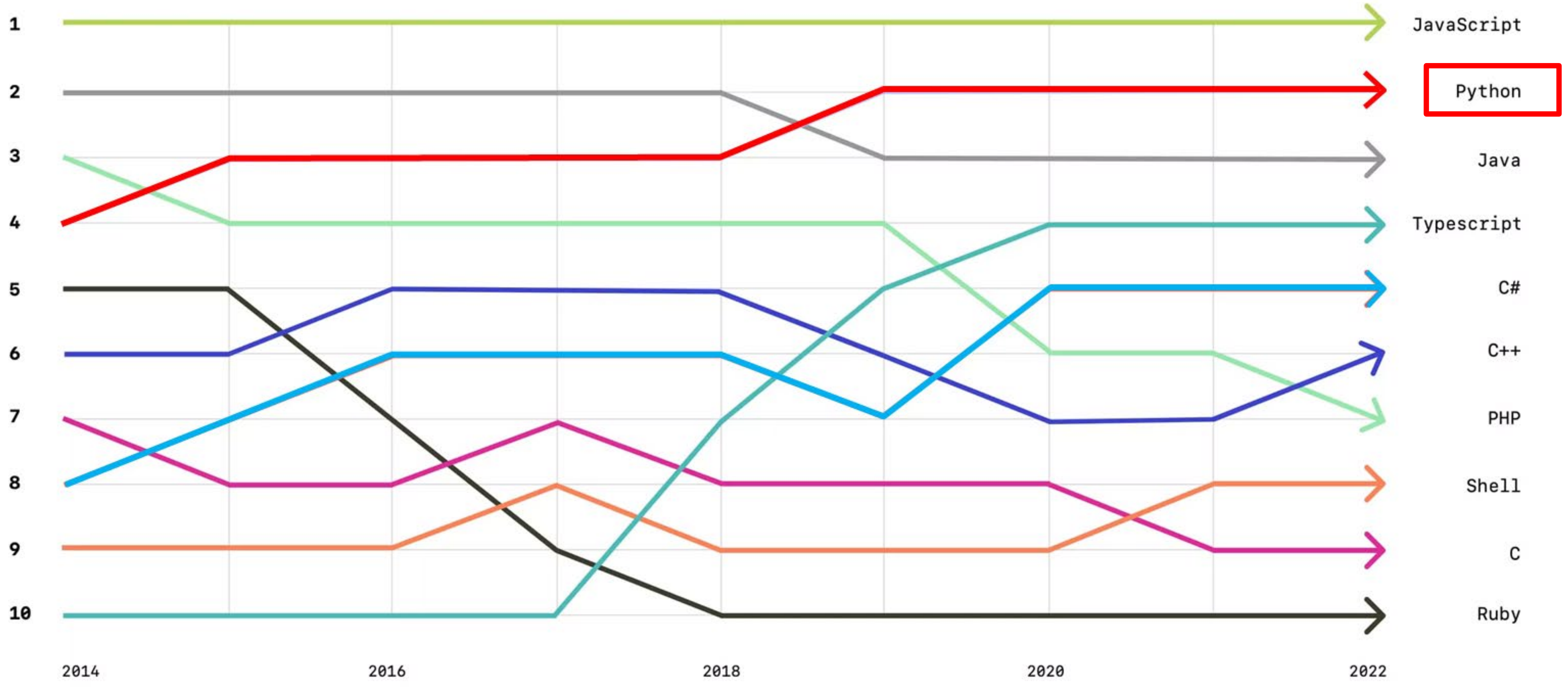
Programming (Creation)

Visual Programming (Logic)

Scripting (Automation)

0.4a What language?

Most used programming languages on GitHub 2014 – 2022:

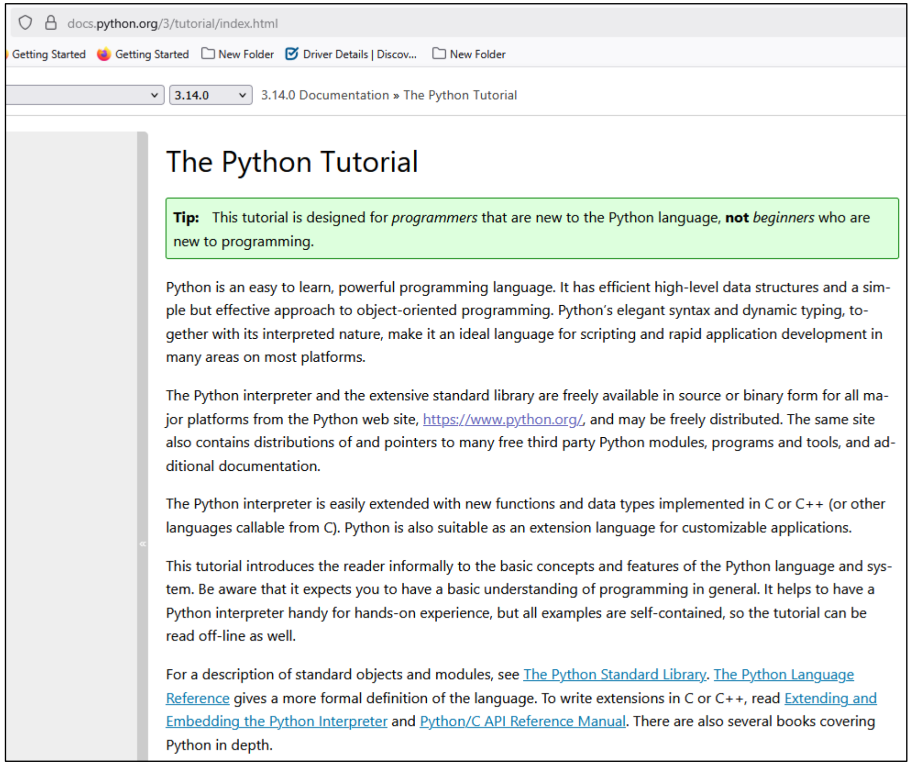


bron:https://solutionshub.epam.com/blog/post/programming-language-popularity-on-github

0.4b What language?

Regarding Rhino’s, Grasshopper’s and Python IDE’s user interfaces, the commands, the functions and the world wide tutorials you can get we will teach this part of DIM in English.

But - no worries – we try to keep it easy to understand. Feel free to aks if you are not able to follow the topics based on language.


docs.python.org/3/tutorial/index.html
Getting Started
Getting Started
New Folder
Driver Details | Discov...
New Folder
3.14.0
3.14.0 Documentation » The Python Tutorial

The Python Tutorial

Tip: This tutorial is designed for *programmers* that are new to the Python language, **not** *beginners* who are new to programming.

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. Be aware that it expects you to have a basic understanding of programming in general. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see The Python Standard Library. The Python Language Reference gives a more formal definition of the language. To write extensions in C or C++, read Extending and Embedding the Python Interpreter and Python/C API Reference Manual. There are also several books covering Python in depth.

https://www.w3schools.com/python/

0.5 A short history

- **1989 – Birth of Python**

  Dutch programmer **Guido van Rossum** started Python during Christmas at CWI **(Centrum Wiskunde & Informatica)** in Amsterdam.

  He wanted a simple, readable scripting language to replace the complex ABC language.

  The name **“Python”** was inspired by the British comedy group **Monty Python**, whose work Guido enjoyed.

- **1991 – Python 1.0**

  The first public release — already included features like functions, exceptions, and modules.

  Motto: **“Simple is better than complex.”**

- **2000 – Python 2.0**

  Introduced Unicode support and garbage collection — made Python useful for web and data applications.

- **2008 – Python 3.0**

  A major clean-up: better text handling, consistent syntax, and modern libraries.

  Not backward compatible with version 2, but set the foundation for today’s Python.

- **2010s – Rise of Data, AI & Design**

  Python became the language of **data science, AI, and engineering automation**, thanks to libraries like NumPy, Pandas, and TensorFlow.

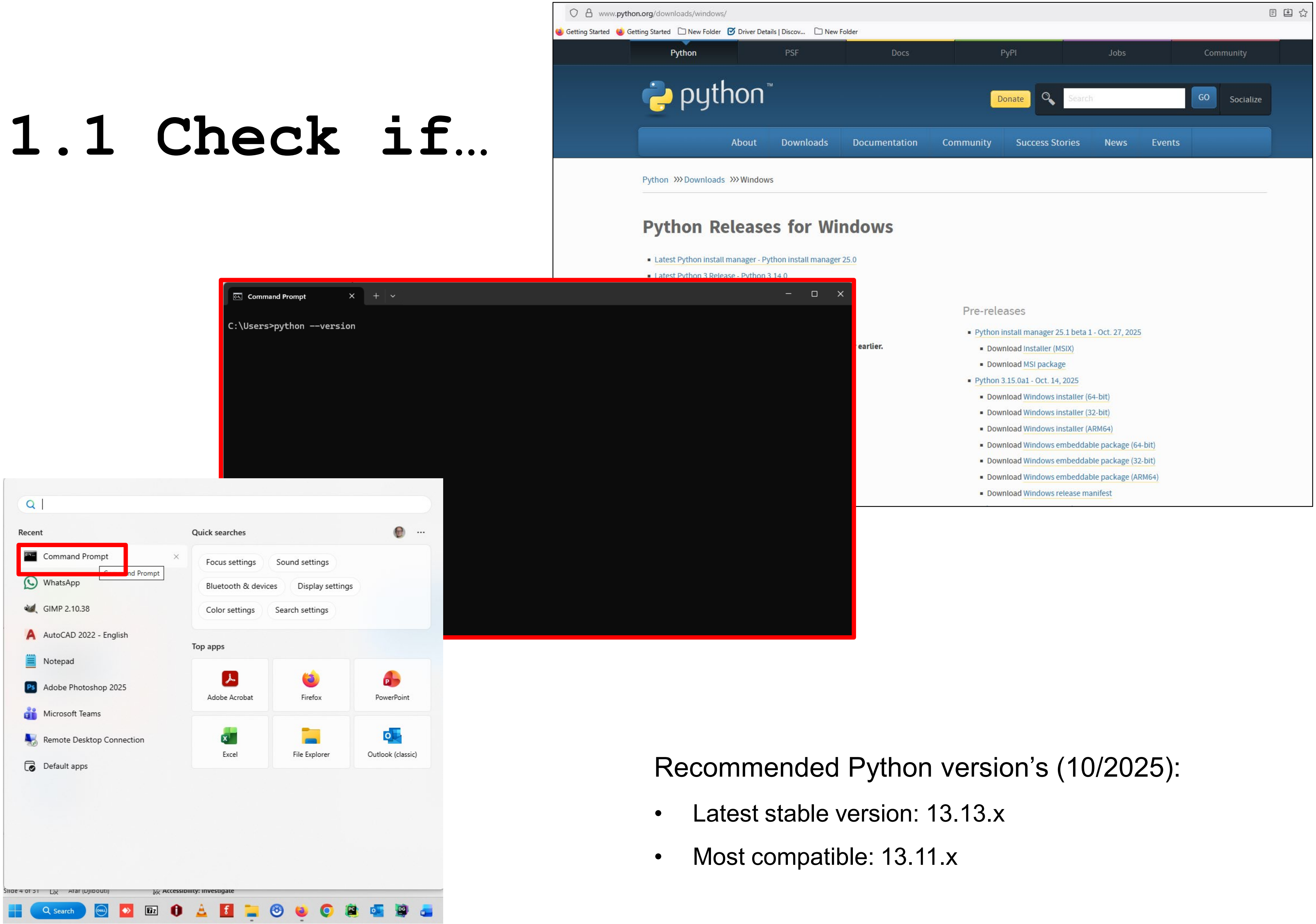
  It also entered the **Built Environment** via Rhino, Revit, and Blender scripting.

- **Today**

  Python is one of the **most used and taught** languages worldwide —

  popular because it’s **easy to read, flexible, and connects to everything** (GIS, (B)IM, Sensors, Web, AI, Robotics).

1.1 Check if…



Python Releases for Windows

- Latest Python install manager - Python install manager 25.0
- Latest Python 3 Release - Python 3.14.0

earlier.

Pre-releases

- Python install manager 25.1 beta 1 - Oct. 27, 2025
  - Download Installer (MSIX)
  - Download MSI package
- Python 3.15.0a1 - Oct. 14, 2025
  - Download Windows installer (64-bit)
  - Download Windows installer (32-bit)
  - Download Windows installer (ARM64)
  - Download Windows embeddable package (64-bit)
  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (ARM64)
  - Download Windows release manifest

```
C:\Users>python --version
```

Recommended Python version’s (10/2025):

- Latest stable version: 13.13.x
- Most compatible: 13.11.x

1.2 How to run Python?

Possible environments to write and run Python codes:

- Text editors (Notebook, Notebook ++, VS Code)
- IDE (PyCharm, Spyder, Thonny, Visual Studio Code)
- Anaconda with Jupyter Notebooks (local)
- Google Collab (Jupyter Notebooks remote / via web interface)
- Installation of Python:

- Standalone in the OS (not used in this course)
- Integrated via the Anaconda environment (not used in this course, but later in VL)
- Integrated via the Revit/Dynamo environment (not used in this course)
- Integrated via the Rhino/Grasshopper environment (used in this course)

1.3 Python References

- Python Documentation by Python Software Foundation

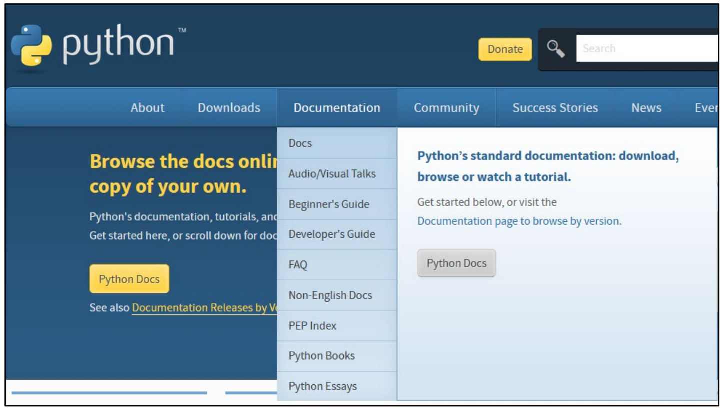
  Python Documentation

- Python Basic Tutorial by tutorialspoint

- Python tutorials by w3schools

- Python in Grasshopper & further

  https://www.youtube.com/watch?v=Ln-ByMyfDy8



Browse the docs onli
copy of your own.

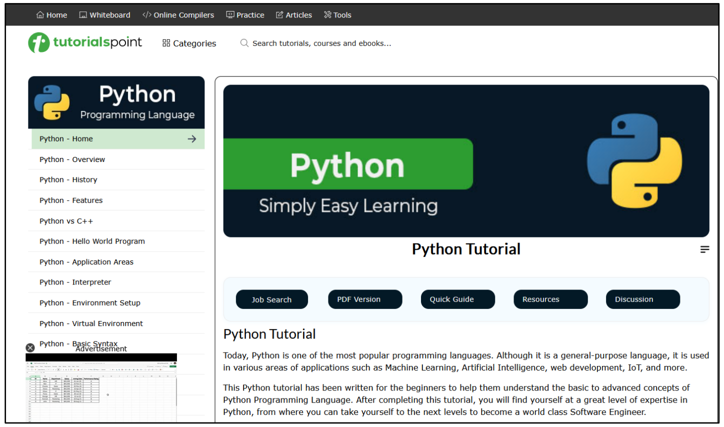
Python's documentation, tutorials, and
Get started here, or scroll down for doc

See also Documentation Releases by V

Python's standard documentation: download, browse or watch a tutorial.

Get started below, or visit the Documentation page to browse by version.

https://www.python.org/doc/



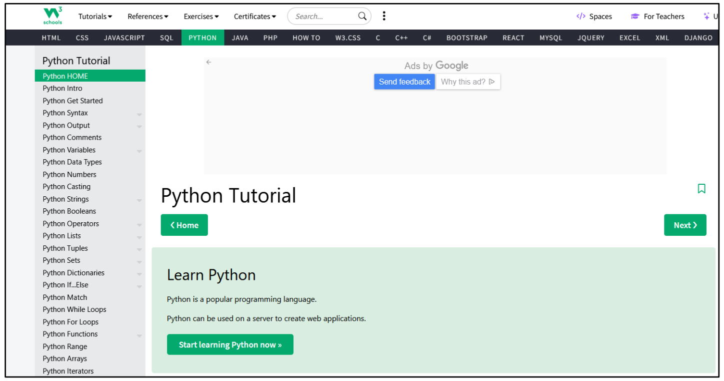
Python Tutorial

Python Tutorial

Today, Python is one of the most popular programming languages. Although it is a general-purpose language, it is used in various areas of applications such as Machine Learning, Artificial Intelligence, web development, IoT, and more.

This Python tutorial has been written for the beginners to help them understand the basic to advanced concepts of Python Programming Language. After completing this tutorial, you will find yourself at a great level of expertise in Python, from where you can take yourself to the next levels to become a world class Software Engineer.

https://www.tutorialspoint.com/python/index.htm



Python Tutorial

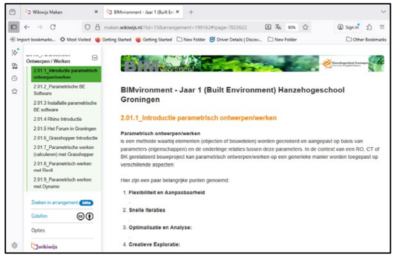
Learn Python

Python is a popular programming language.

Python can be used on a server to create web applications.

https://www.w3schools.com/python/

1.5 Let's start!



BIMvironment - Jaar 1 (Built Environment) Hanzehogeschool Groningen

2.01.1_Introductie parametrisch ontwerpen/werken

Parametrisch ontwerpen/werken
is een methode waarbij elementen (objecten of bouwdelen) worden gecreëerd en aangepast op basis van parameters (eigenschappen) en de onderlinge relaties tussen deze parameters. In de context van een RO, CT of BK gerelateerd bouwproject kan parametrisch ontwerpen/werken op een generieke manier worden toegepast op verschillende aspecten:

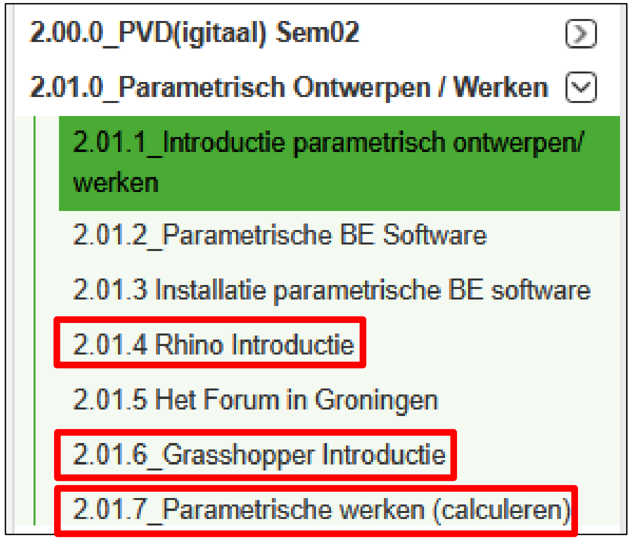
Hier zijn een paar belangrijke punten genoemd:

1. **Flexibiliteit en Aanpasbaarheid**
2. **Snelle Iteraties**
3. **Optimalisatie en Analyse:**
4. **Creatieve Exploratie:**

Ontwerpen / Werken
2.01.1_Introductie parametrisch ontwerpen/werken
2.01.2_Parametrische BE Software
2.01.3 Installatie parametrische BE software
2.01.4 Rhino Introductie
2.01.5 Het Forum in Groningen
2.01.6_Grasshopper Introductie
2.01.7_Parametrische werken (calculeren) met Grasshopper
2.01.8_Parametrisch werken met Revit
2.01.9_Parametrisch werken met Dynamo
Zoeken in arrangement bèta
Colofon
Opties
wikiwijs

Recap Rhino & Grasshoper

- Recap the PVD_02 Grasshopper / Rhino exercises on WikiWijs.



2.00.0_PVD(igitaal) Sem02

2.01.0_Parametrisch Ontwerpen / Werken

2.01.1_Introductie parametrisch ontwerpen/werken

2.01.2_Parametrische BE Software

2.01.3 Installatie parametrische BE software

2.01.4 Rhino Introductie

2.01.5 Het Forum in Groningen

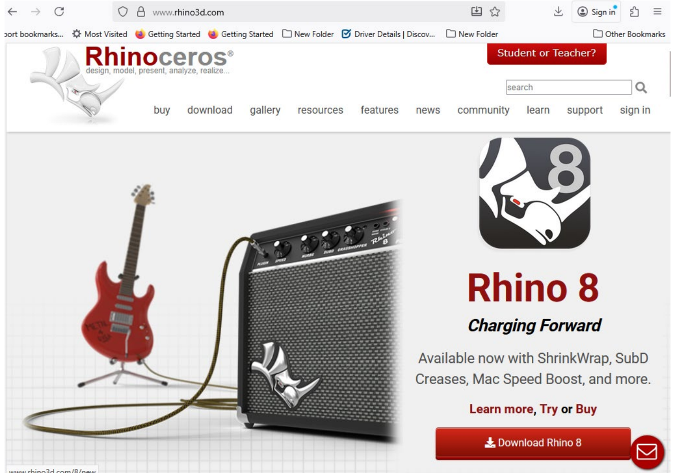
2.01.6_Grasshopper Introductie

2.01.7_Parametrische werken (calculeren)

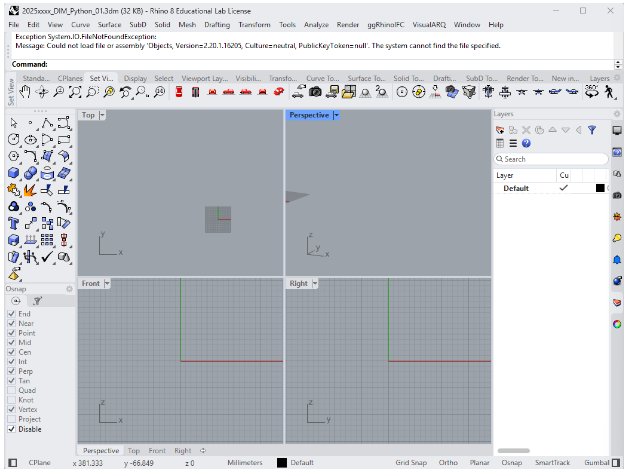
The Wikiwijs PVD_02 environment

Open Rhino 8 with Grasshopper:

- Create an empty project (large scale in mm) in Rhino 8 and save it.
- Start the included Grasshopper visual programming environment.



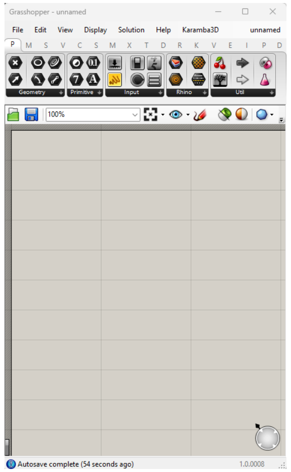
bron: www.Rhino3d.com



2025xxxx_DIM_Python_01.3dm (32 KB) - Rhino 8 Educational Lab License

```
Exception System.IO.FileNotFoundException:
Message: Could not load file or assembly 'Objects, Version=2.20.1.16205, Culture=neutral, PublicKeyToken=null'. The system cannot find the file specified.
Command:
```

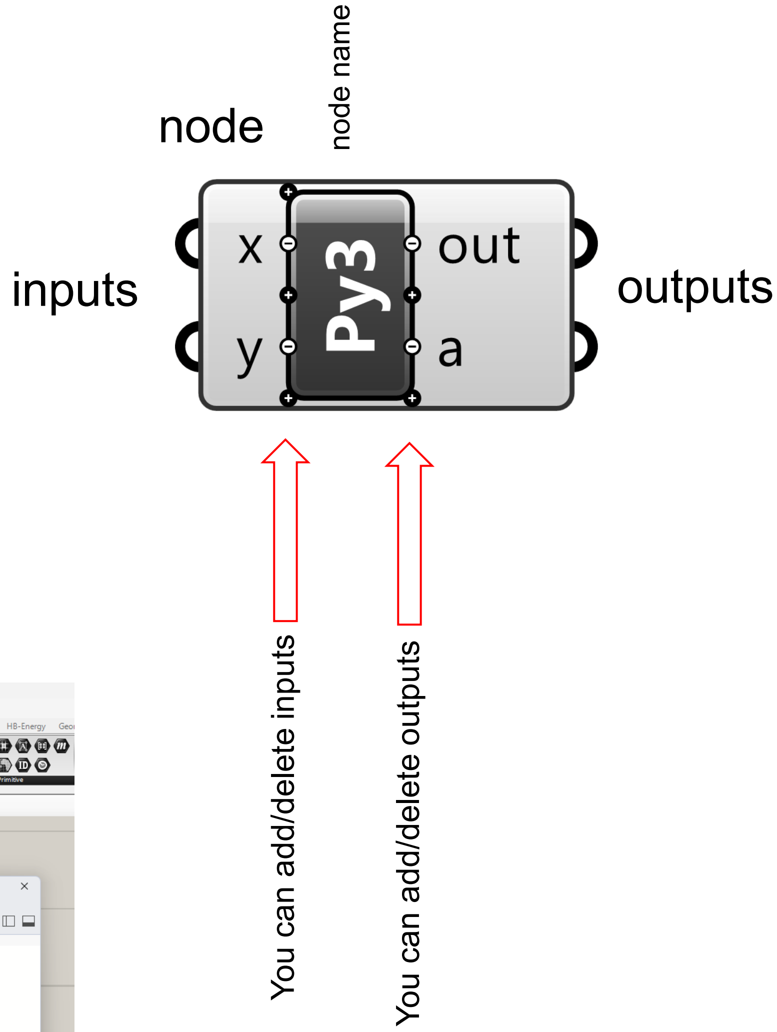
Rhino UI



Grasshopper UI

1.6 The Python3 node

The python 3 node



node

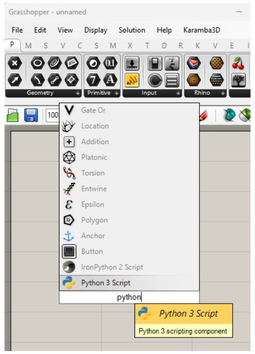
node name

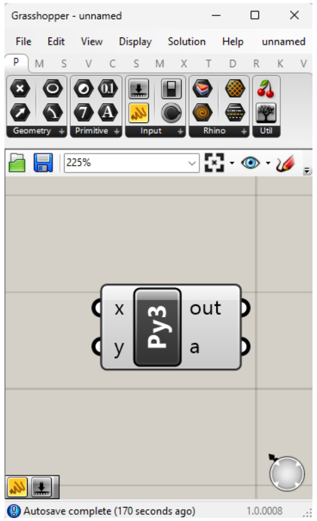
inputs

outputs

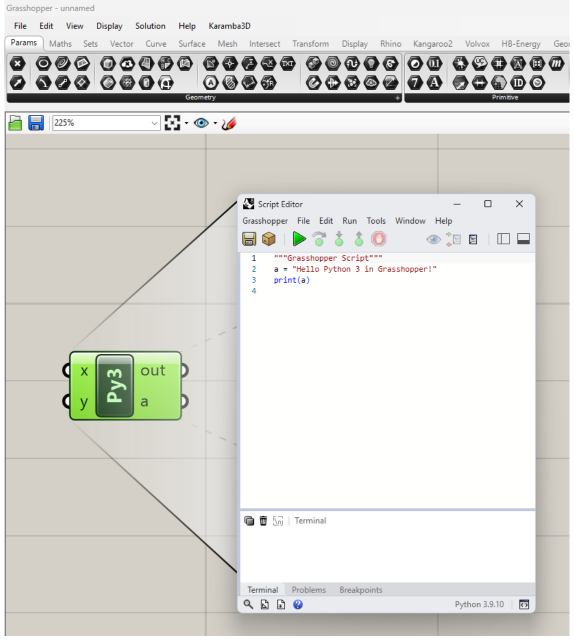
You can add/delete inputs

You can add/delete outputs





Autosave complete (170 seconds ago)



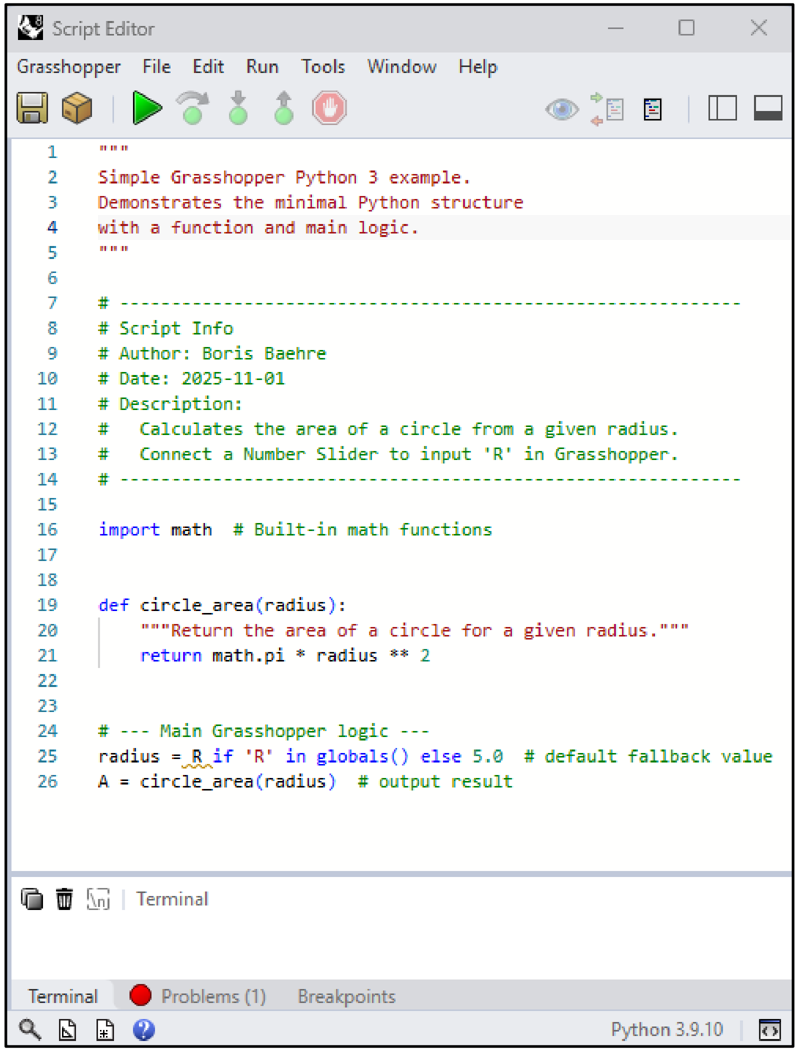
```
"""Grasshopper Script"""
a = "Hello Python 3 in Grasshopper!"
print(a)
```

2.1 Python code structure:

A typical Python code structure includes the following elements:

- Shebang (Optional, not used in Grasshopper)
- Docstring (optional, but mendatory in DIM)
- Imports (in this example: math)
- Functions (indicated by the key expression ‘def’)
- Classes (optional and not used in this example)
- Main Program Logic (mentioned as a comment #)
- Error Handling (optional and not used in this example)

This structure helps organize code
for readability and maintainability.

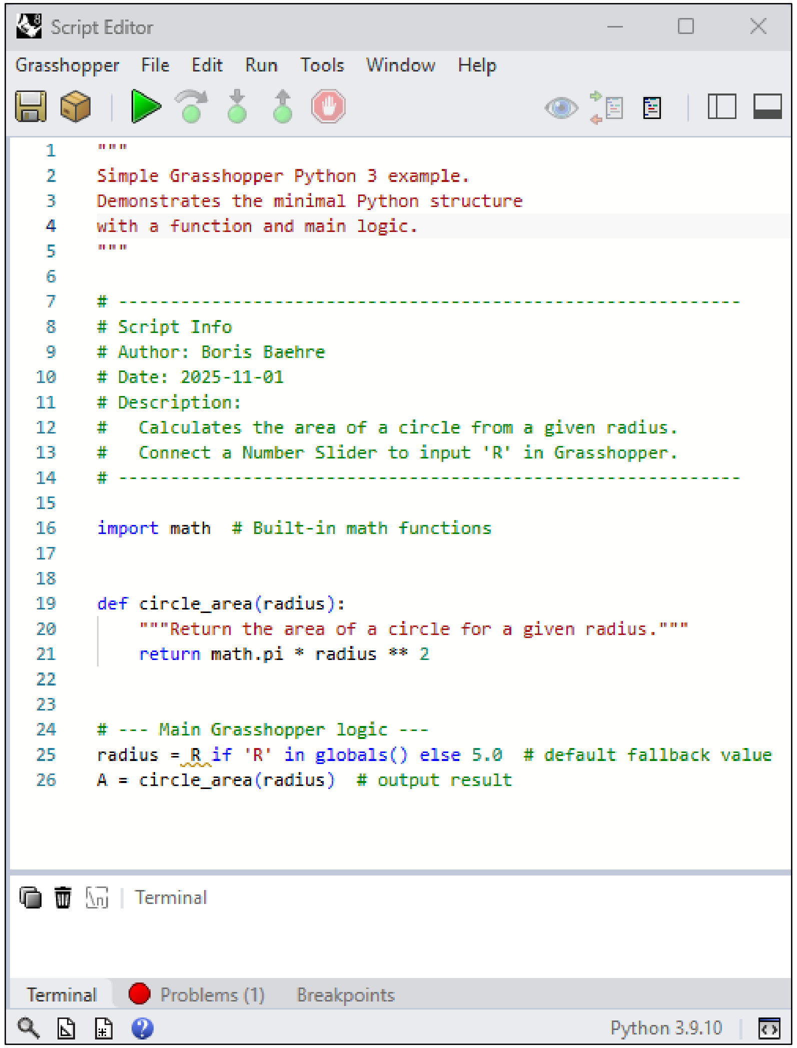


```python
"""
Simple Grasshopper Python 3 example.
Demonstrates the minimal Python structure
with a function and main logic.
"""

# ------------------------------------------------------------
# Script Info
# Author: Boris Baehre
# Date: 2025-11-01
# Description:
#   Calculates the area of a circle from a given radius.
#   Connect a Number Slider to input 'R' in Grasshopper.
# ------------------------------------------------------------

import math  # Built-in math functions


def circle_area(radius):
    """Return the area of a circle for a given radius."""
    return math.pi * radius ** 2


# --- Main Grasshopper logic ---
radius = R if 'R' in globals() else 5.0  # default fallback value
A = circle_area(radius)  # output result
```

Let's start!

3.0 Basic syntax

- Python syntax refers to the rules that define how code is written and structured.



```python
"""
Simple Grasshopper Python 3 example.
Demonstrates the minimal Python structure
with a function and main logic.
"""

# -------------------------------------------------------------
# Script Info
# Author: Boris Baehre
# Date: 2025-11-01
# Description:
#   Calculates the area of a circle from a given radius.
#   Connect a Number Slider to input 'R' in Grasshopper.
# -------------------------------------------------------------

import math  # Built-in math functions


def circle_area(radius):
    """Return the area of a circle for a given radius."""
    return math.pi * radius ** 2


# --- Main Grasshopper logic ---
radius = R if 'R' in globals() else 5.0  # default fallback value
A = circle_area(radius)  # output result
```

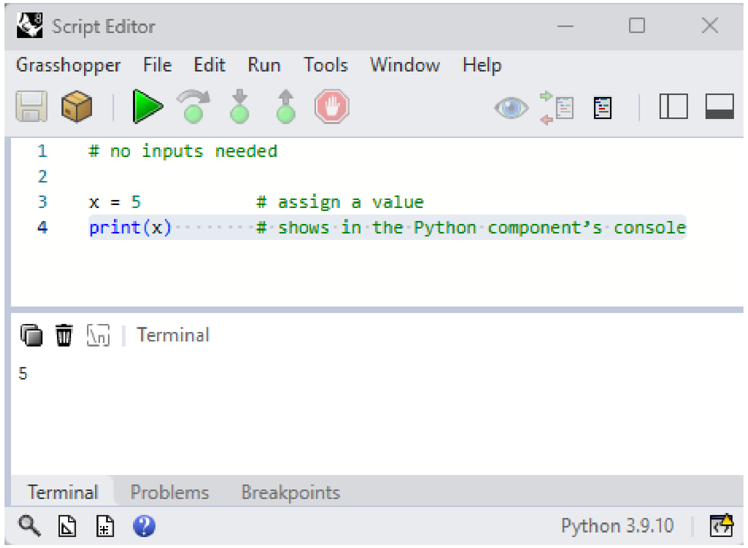
Key Syntax Rules to Remember

1. Indentation matters - Blocks of code use spaces, not braces {}
2. Case-sensitive - **Name** is not the same as **name**
3. No semicolons needed - One command per line is default.
4. Everything is an object - Numbers, strings, lists, functions - all are objects.
5. Whitespace is meaningful - Tabs/spaces structure the logic - **they are not decoration!**

3.1 Variables

Example:

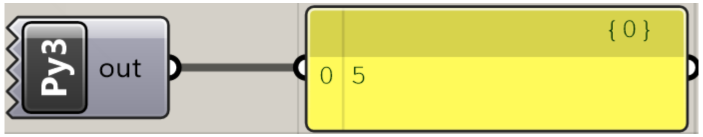
- In the following example, x stores an integer.
  The print() function is used to display the values of x.

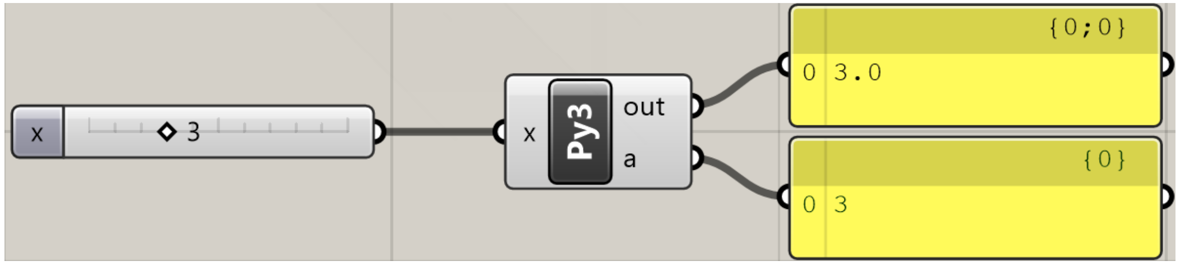

Script Editor
Grasshopper File Edit Run Tools Window Help

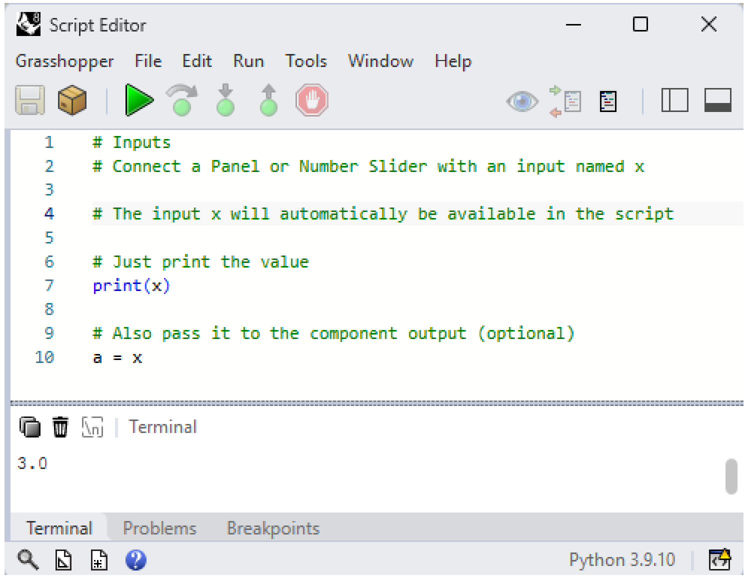
```
# no inputs needed

x = 5               # assign a value
print(x)            # shows in the Python component’s console
```

```
5
```

Terminal Problems Breakpoints
Python 3.9.10






Script Editor
Grasshopper File Edit Run Tools Window Help

```
# Inputs
# Connect a Panel or Number Slider with an input named x

# The input x will automatically be available in the script

# Just print the value
print(x)

# Also pass it to the component output (optional)
a = x
```

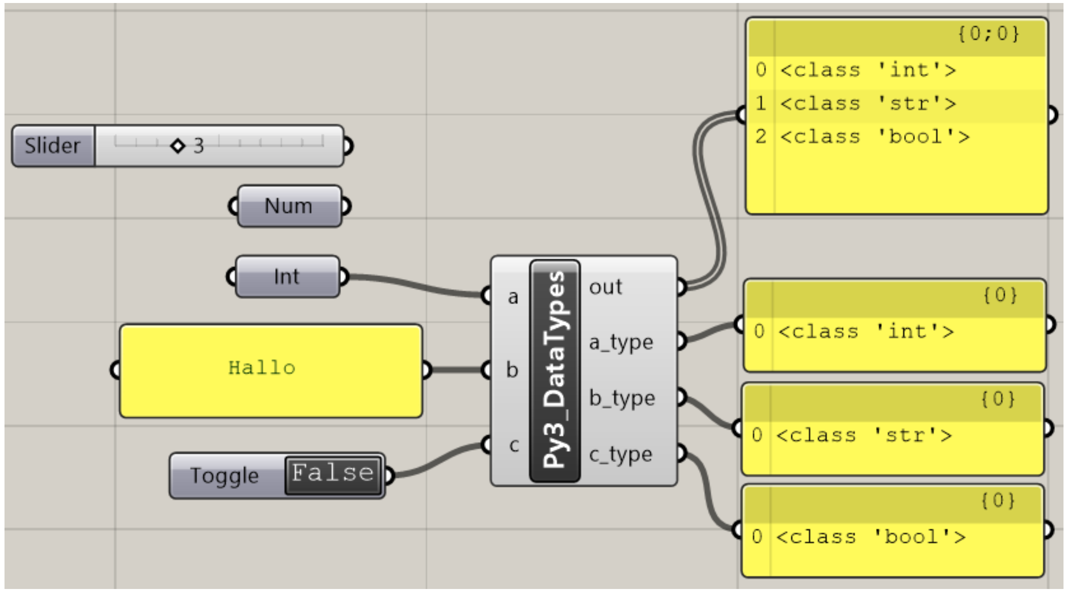
```
3.0
```

Terminal Problems Breakpoints
Python 3.9.10

- The variable a is introduced to create an output where the value of x is passed to

3.2 The type of data

- The `type()` function is used to check the type of the variable, showing ‘a’ is an integer, ‘b’ is a string and ‘c’ is a boolean.

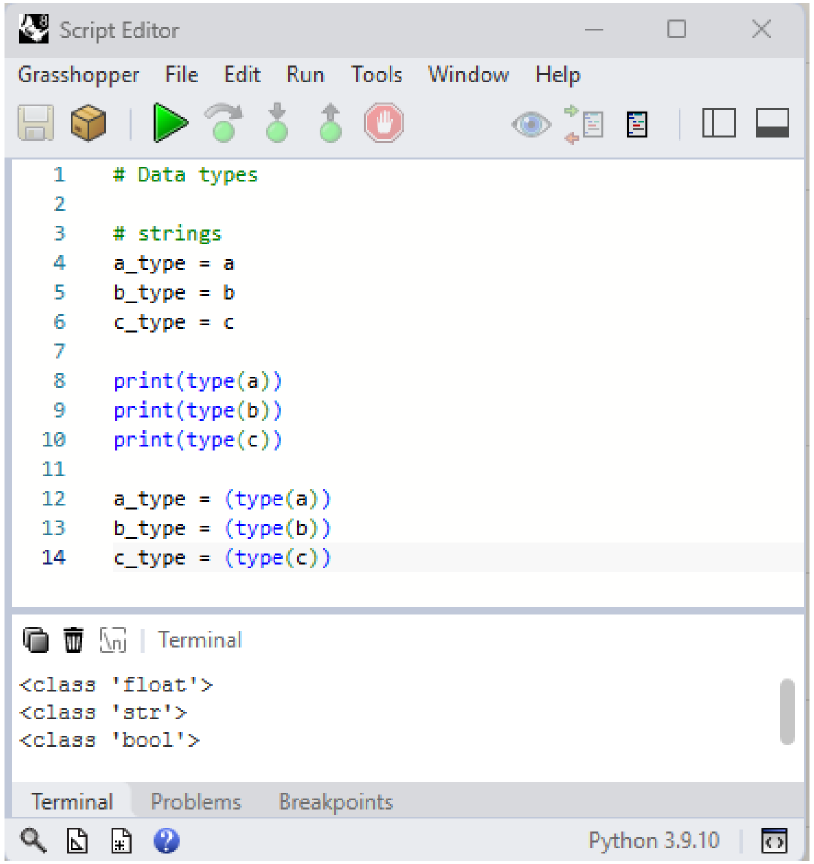


```
0 <class 'int'>
1 <class 'str'>
2 <class 'bool'>
```

```
0 <class 'int'>
```

```
0 <class 'str'>
```

```
0 <class 'bool'>
```

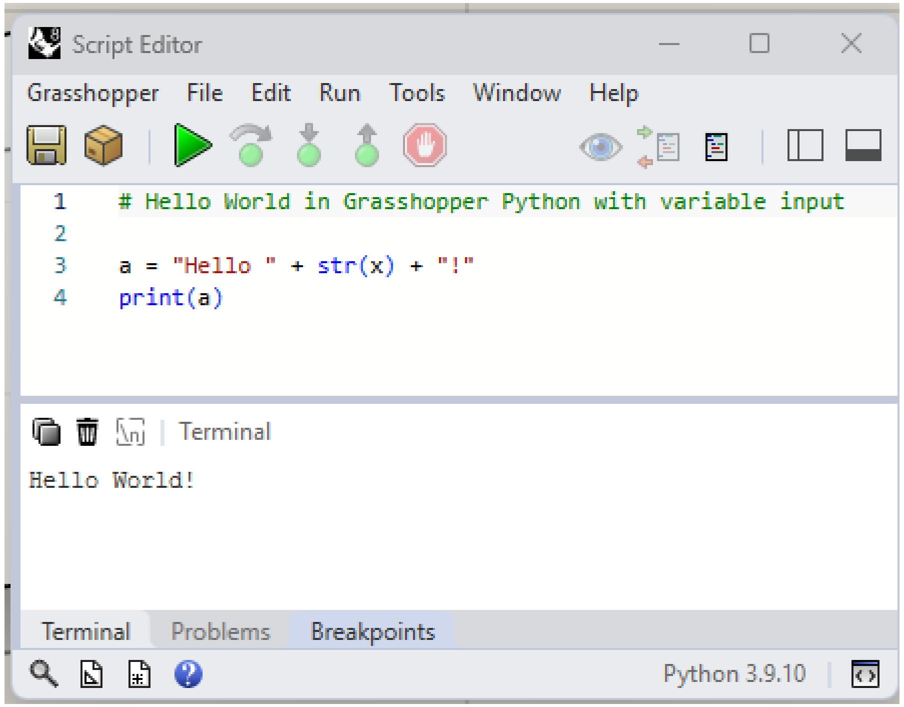


```
# Data types

# strings
a_type = a
b_type = b
c_type = c

print(type(a))
print(type(b))
print(type(c))

a_type = (type(a))
b_type = (type(b))
c_type = (type(c))
```

```
<class 'float'>
<class 'str'>
<class 'bool'>
```

- An ‘integer’ is a whole number, positive or negative, without a decimal point (e.g., 5, -10, 0).
- A ‘string’ is a text ….
- A ‘float” is a number that has a decimal point (e.g., 5.0, -10.75, 0.3).

3.3 Hello World



Script Editor
Grasshopper File Edit Run Tools Window Help

```
# Hello World in Grasshopper Python with variable input

a = "Hello " + str(x) + "!"
print(a)
```

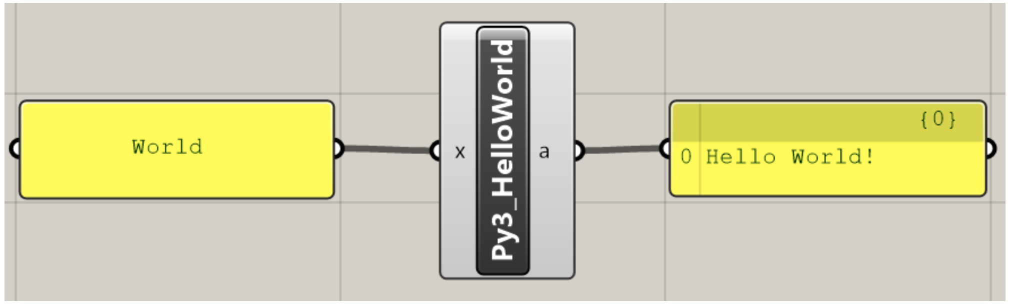
```
Hello World!
```

Terminal Problems Breakpoints
Python 3.9.10

The **"Hello, World!"** story is basically the tradition of writing the very first, simplest program when learning a new programming language.

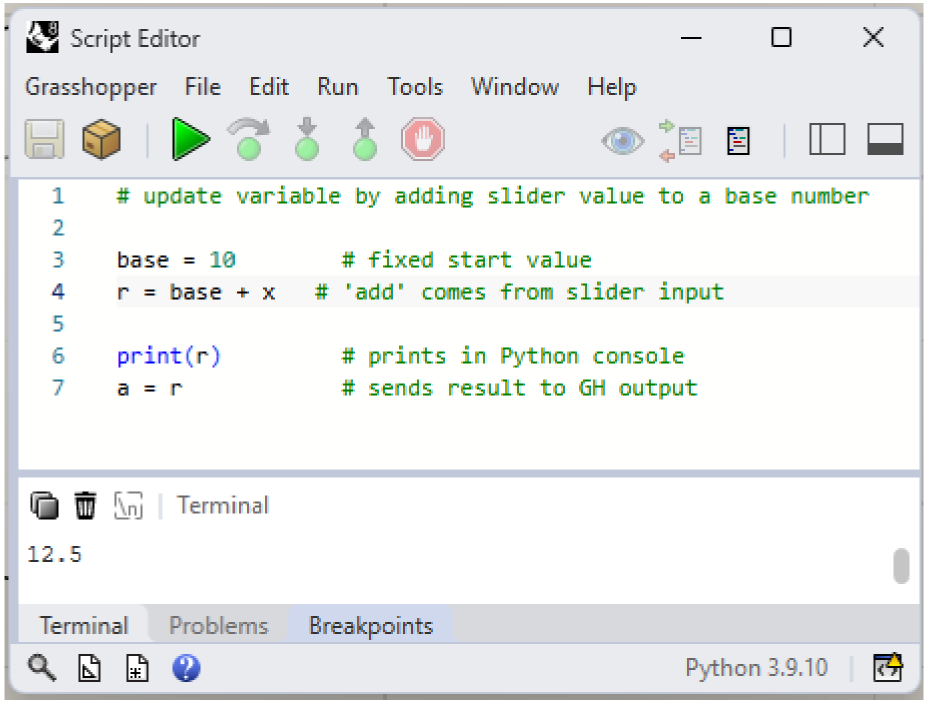
- It started in **1972** in the book *The C Programming Language* by **Brian Kernighan & Dennis Ritchie*.
- Its purpose was only to show how to display text on the screen – to prove that the compiler, language, and computer setup all work.

Since then, almost every tutorial for any language begins with printing **"Hello, World!"** as a friendly first step.



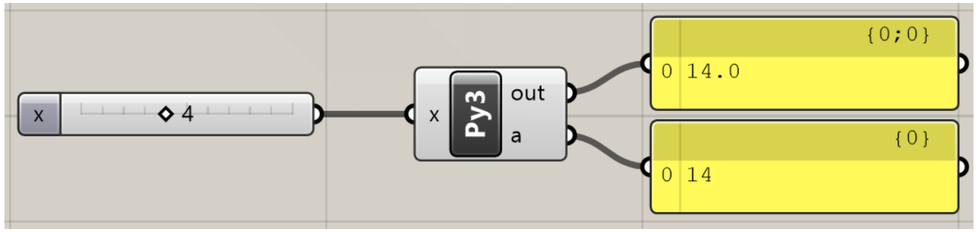
3.4 Updataing variables

- Here, the value of r is updated by adding 2.5 to its current value, and the new value is printed and pushed to output a


Script Editor
Grasshopper File Edit Run Tools Window Help
Terminal
Terminal Problems Breakpoints
Python 3.9.10

```
# update variable by adding slider value to a base number

base = 10         # fixed start value
r = base + x   # 'add' comes from slider input

print(r)          # prints in Python console
a = r             # sends result to GH output
```

```
12.5
```



3.5 Variable Name

A Python identifier is a name used to identify a variable, function, class, module or other object.

- An identifier can have any letter from A to Z (or a to z), or an underscore or digits (0 to 9).
- Python does not allow punctuation characters (e.g. @, $, #, %) within identifiers.
- Python is a case sensitive programming language.

Thus, **Manpower** and **manpower** are two different identifiers in Python.

Reserved words

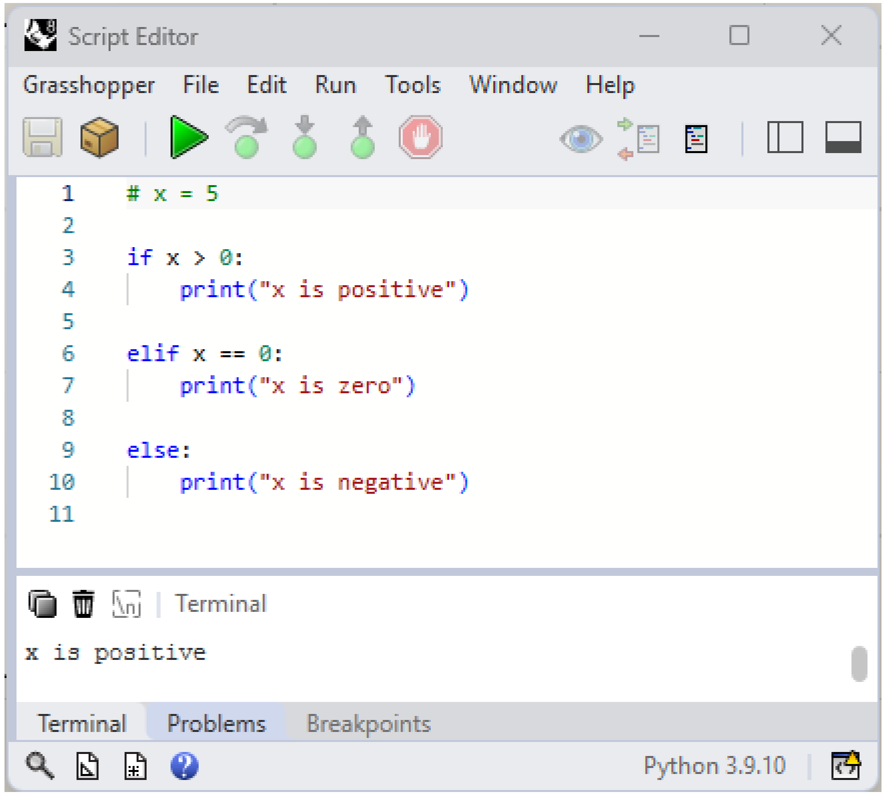
- Some words in Python are reserved, not able to be used as constants or variables or any other identifier names:

  and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield.

3.6 Indentation

Python uses indentation (spaces or tabs) to define blocks of code, instead of braces {} like in other languages.
All statements inside a block must have the same amount of indentation.

- Lines with the same indentation level belong to the same block of code.
- The number of spaces used is up to you, but they must be consistent within the block.
- Returning to the previous indentation level ends the block.
- Empty lines are ignored.
- Everything after # and between “” is a comment and ignored by Python.

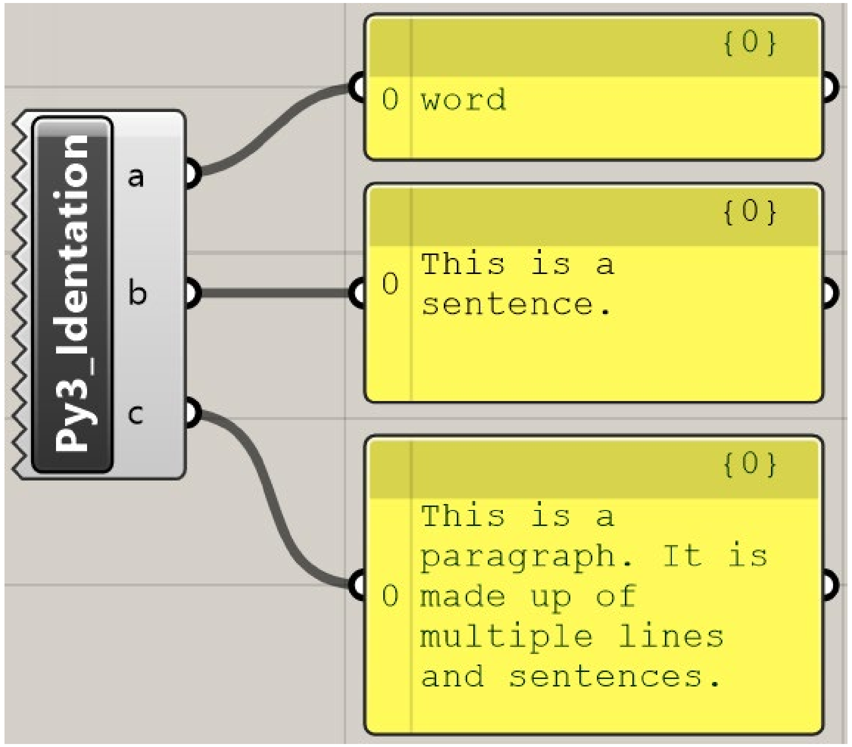


```
# x = 5

if x > 0:
    print("x is positive")

elif x == 0:
    print("x is zero")

else:
    print("x is negative")

```

```
x is positive
```

3.7 Quotation

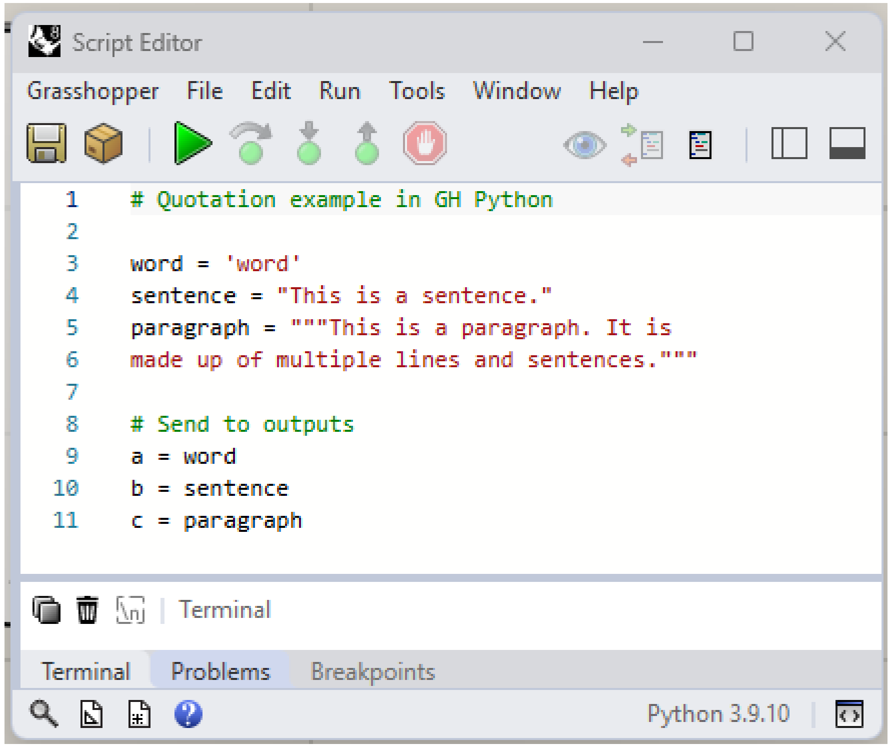
Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string. The triple quotes are used to span the string across multiple lines.



```
{0}
0 word
```

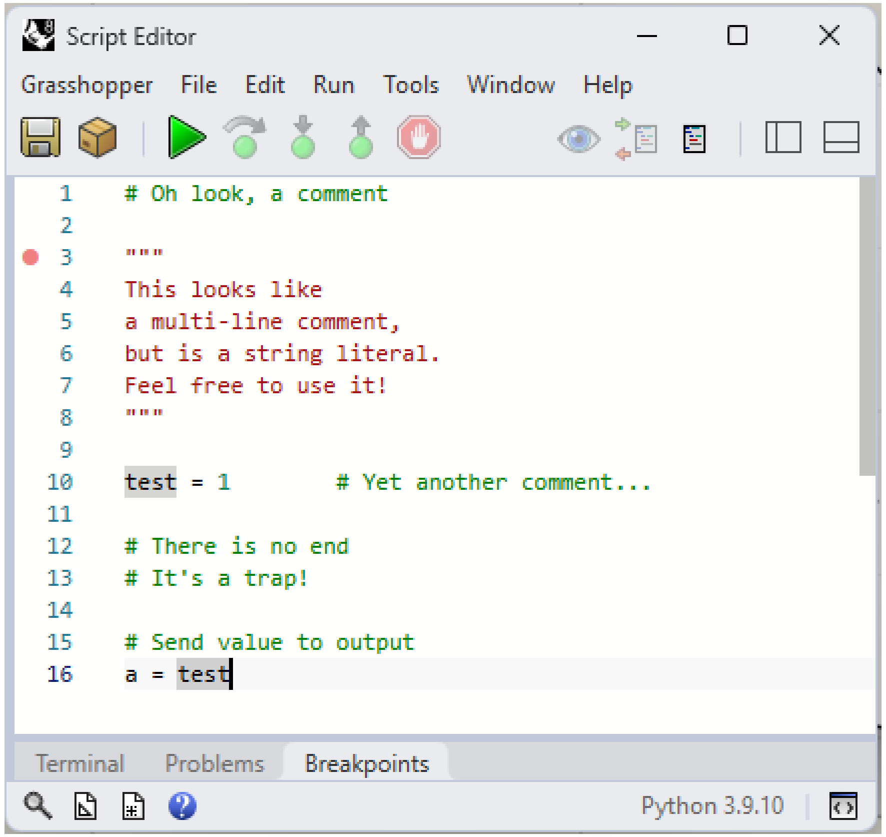
```
{0}
0 This is a
  sentence.
```

```
{0}
  This is a
  paragraph. It is
0 made up of
  multiple lines
  and sentences.
```


Script Editor
Grasshopper File Edit Run Tools Window Help
Terminal
Terminal Problems Breakpoints
Python 3.9.10

```python
# Quotation example in GH Python

word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""

# Send to outputs
a = word
b = sentence
c = paragraph
```

3.8 Comments


Script Editor
Grasshopper File Edit Run Tools Window Help
Terminal Problems Breakpoints
Python 3.9.10

```python
# Oh look, a comment

"""
This looks like
a multi-line comment,
but is a string literal.
Feel free to use it!
"""

test = 1        # Yet another comment...

# There is no end
# It's a trap!

# Send value to output
a = test
```

Comments in Python start with the hash caracter (#), for each of the lines of the comment.

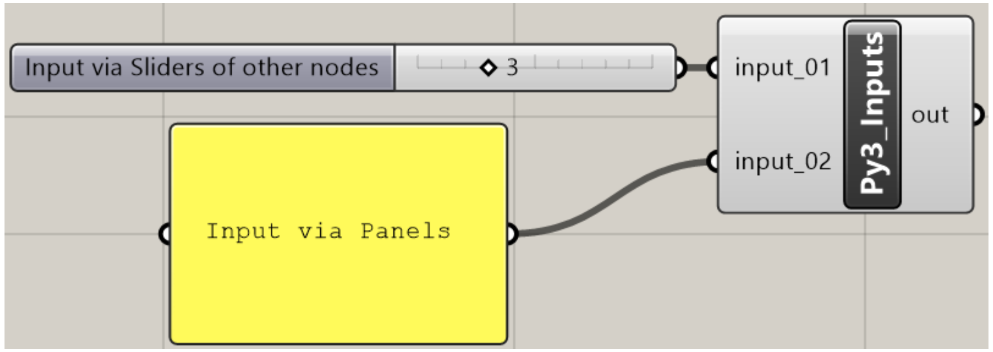
A comment may appear at the start of a line or following whitespace or code, but not within a string literal.

As good practice, comment as much as possible your code (very useful for the future you).

3.9 Inputs

Inputs offer very useful functionality in Puthon and are normally provided with the function input(). Take care: This functionality is slightly different while using the Rhino/Grasshopper/Python environments. The inputs need to be defined by inpuit nodes such as int, num, number-slider or bool and are immediately running – once connected with the python node.

Check carefully before connecting them – in case of errors they might freeze Grasshopper.



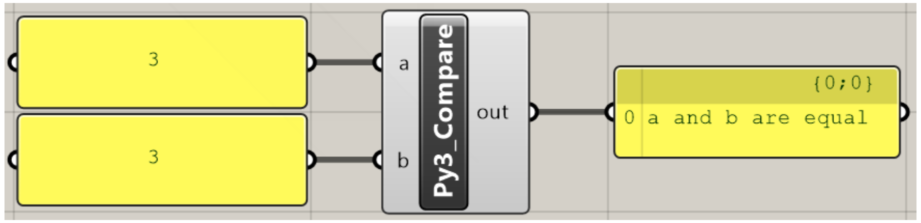
4.0 Conditional Statements

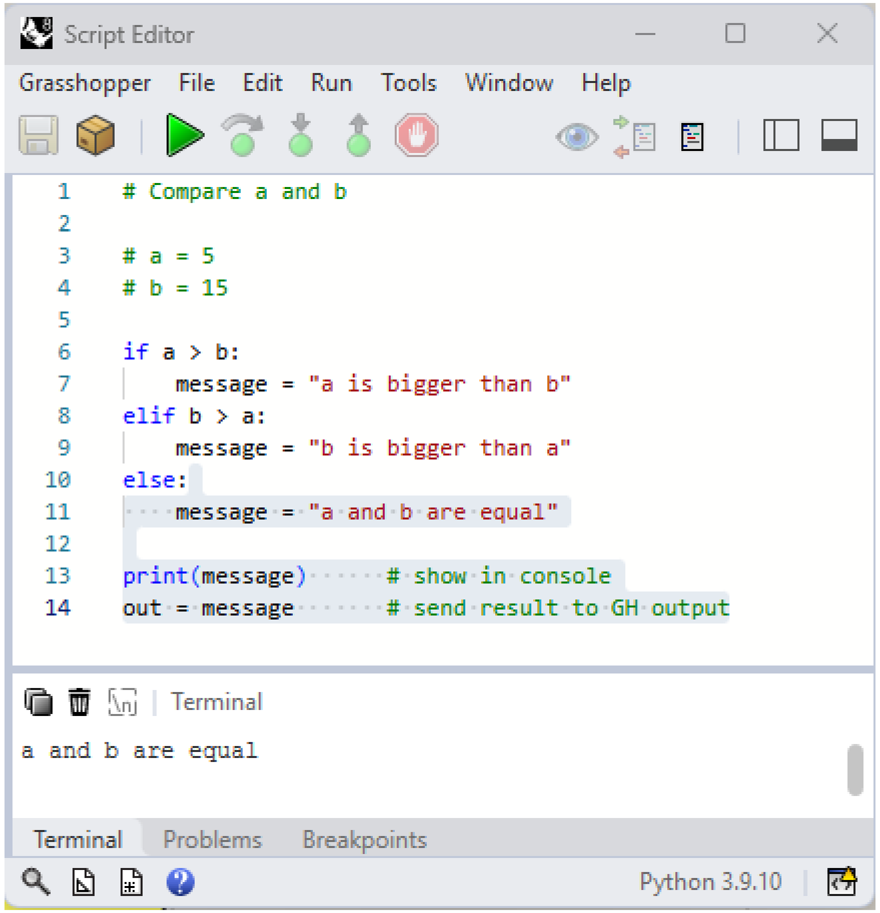
Conditional statements allow you to control the flow of your program by executing certain blocks of code only if specific conditions are met. In Python, you use if, elif, and else to handle decision-making.

if: Runs a block of code if the given condition is True.

elif: (short for "else if") Allows you to check additional conditions if the first if condition is False.

else: Executes a block of code when none of the previous conditions are True.





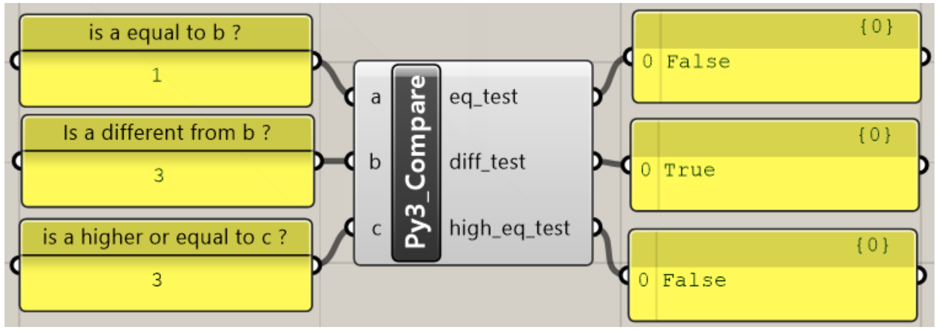
```python
# Compare a and b

# a = 5
# b = 15

if a > b:
    message = "a is bigger than b"
elif b > a:
    message = "b is bigger than a"
else:
    message = "a and b are equal"

print(message)     # show in console
out = message      # send result to GH output
```

```
a and b are equal
```

4.1 Comparison Operators

These operators compare the values on either sides of them and decide the relation among them.

They are also called Relational operators.

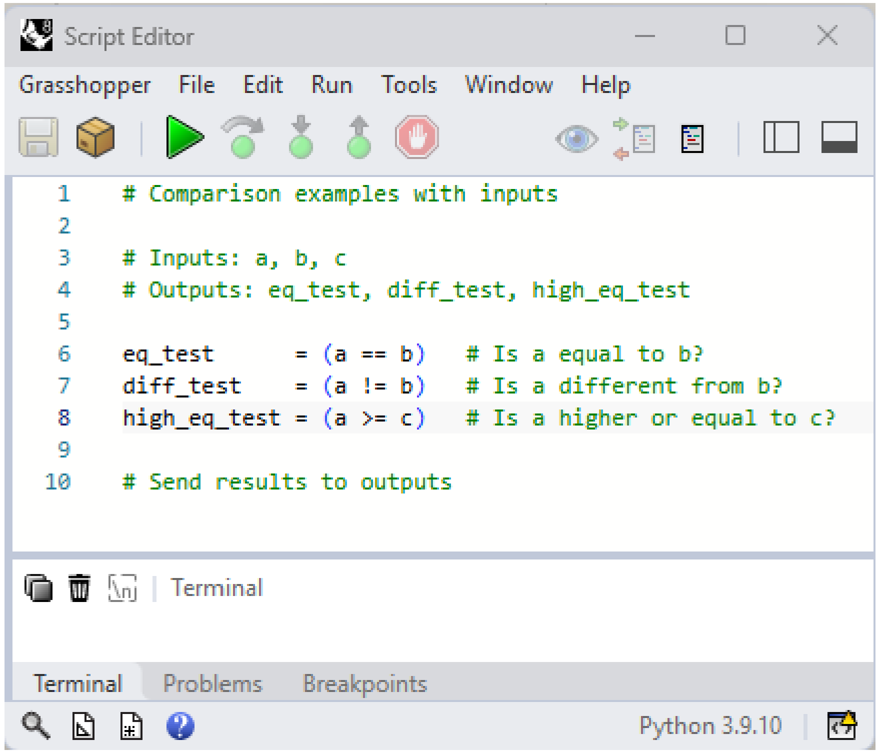


is a equal to b ?
1

Is a different from b ?
3

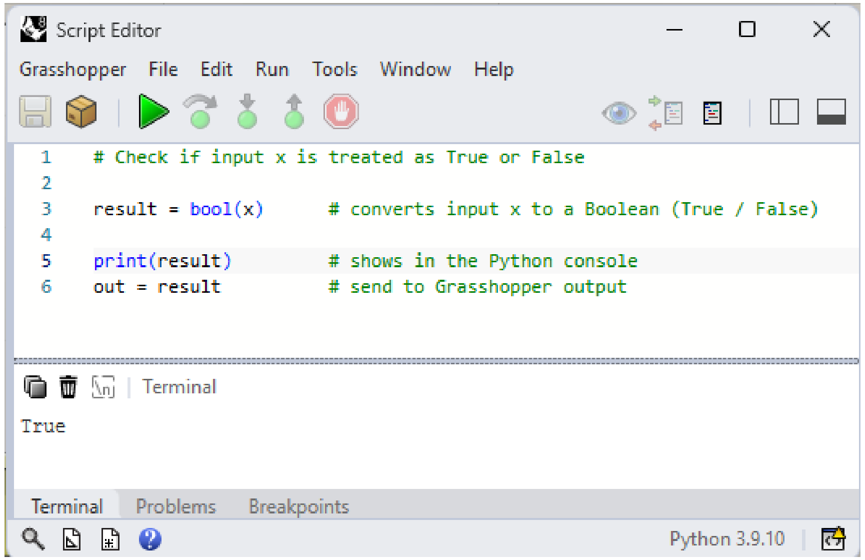
is a higher or equal to c ?
3

```
{0}
0 False
{0}
0 True
{0}
0 False
```



```
# Comparison examples with inputs

# Inputs: a, b, c
# Outputs: eq_test, diff_test, high_eq_test

eq_test      = (a == b)   # Is a equal to b?
diff_test    = (a != b)   # Is a different from b?
high_eq_test = (a >= c)   # Is a higher or equal to c?

# Send results to outputs
```

4.2 Bool


Script Editor
Grasshopper File Edit Run Tools Window Help
Terminal
Terminal Problems Breakpoints
Python 3.9.10

```
# Check if input x is treated as True or False

result = bool(x)       # converts input x to a Boolean (True / False)

print(result)          # shows in the Python console
out = result           # send to Grasshopper output
```

```
True
```

Boolean variables hold a True or False value.

In addition to those two values, python treats the following as **True:**

True

any non-zero numeric value

a non-empty list

a non-empty tuple

a non-empty dictionary

a non-empty string

In contrast, the following are treated as **False**:

False

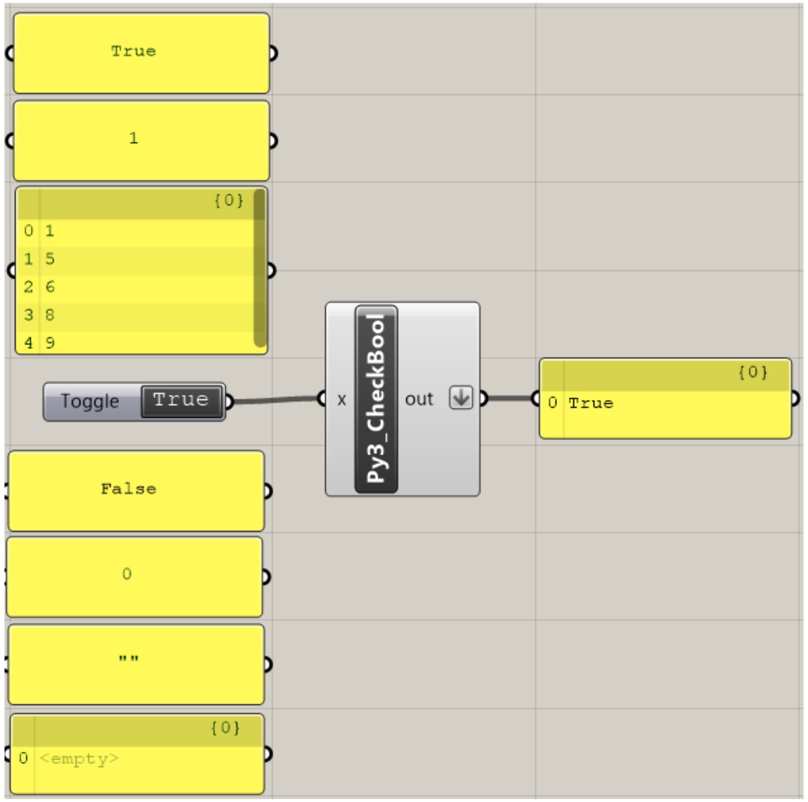
0 (int) or 0.0 (float)

empty list: []

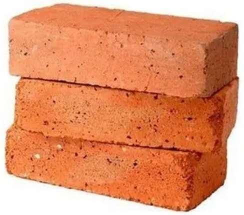
empty dictionary: {}

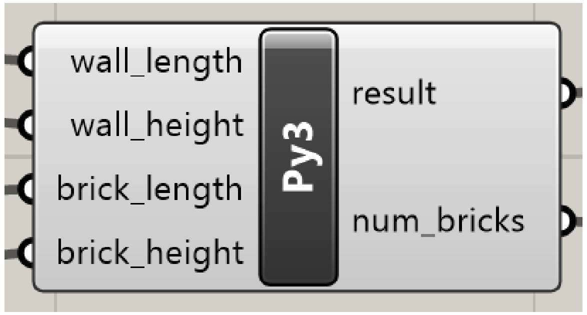
empty string: ''

empty pointer: None



4.2 Homework





The Simple Wall Brick Calculator (in mm)”

Goal

Use Python in Grasshopper to calculate how many rectangular bricks are needed to build a wall – and to present the result neatly using text formatting.

Tips:

- Use **variables** and **data types** (numbers, strings)
- Do the **basic calculations inside the code**
- Use **if-statements**
- Use **inputs and outputs** in the GhPython component

Optional: Try to practice **f-strings** and number **formatting** so that the result looks more clean.

…