>>> Python

a crash course on Grasshopper Python components

>>> Day 1: intro to Python

>>> What is Python?

>>> What is Python?

open source; interpreted; object-oriented; high-level; dynamically syntax; programming language

>>> What is IronPython?

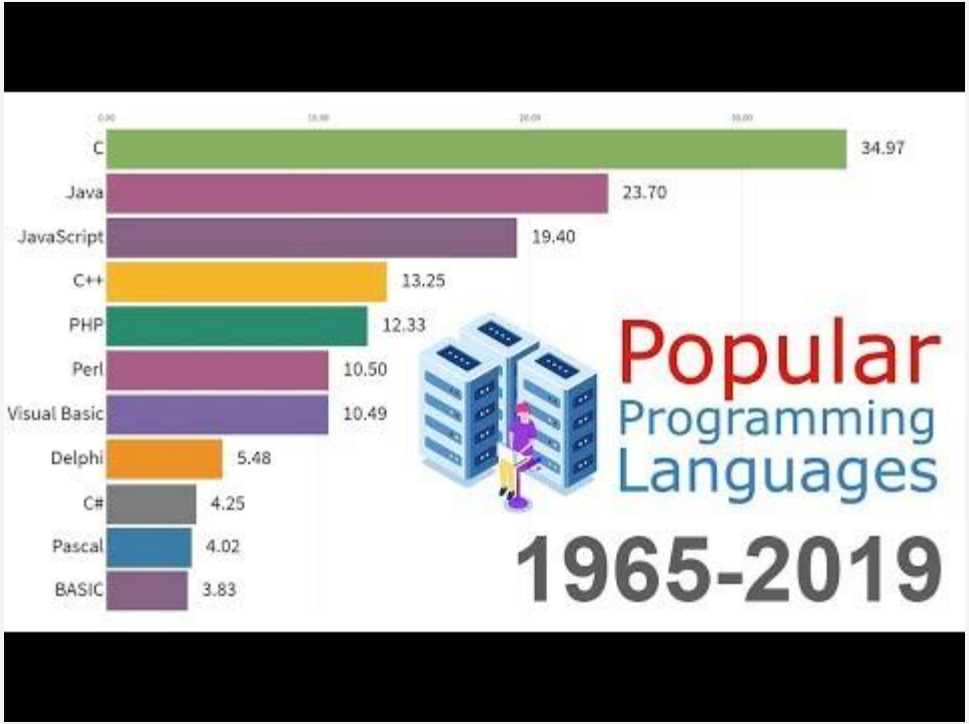
Simplified answer:

- An open-source implementation of Python designed for the .NET and Mono platforms, developed by Microsoft.
- The implementation allows Python to use the Common Language Runtime (clr) to talk directly to other .NET applications (e.g. C#, F#, VB.Net) and libraries.
- There is discrepancy between IronPython and Python (CPython).

>>> What is IronPython?

>>> Why Python?

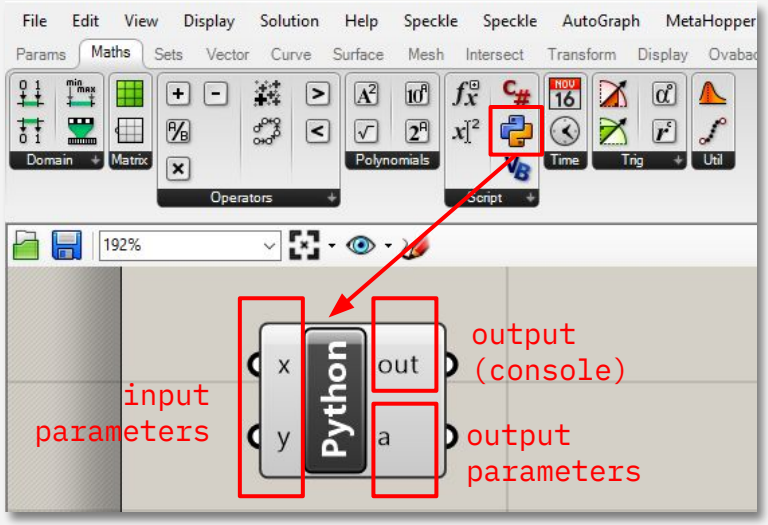
- Simple syntax and easy to learn
- Versatility (endless opportunities)
- Open source community and resources
- Value and demand



>>> Why Python?

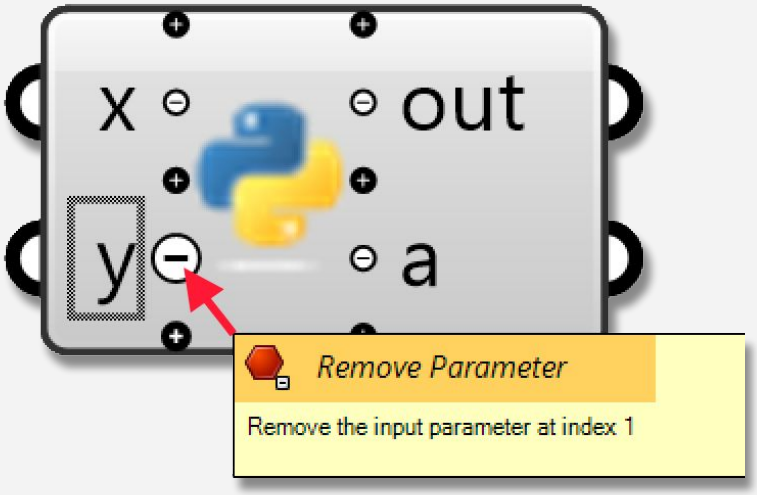
>>> Writing IronPython code

on Grasshopper environment



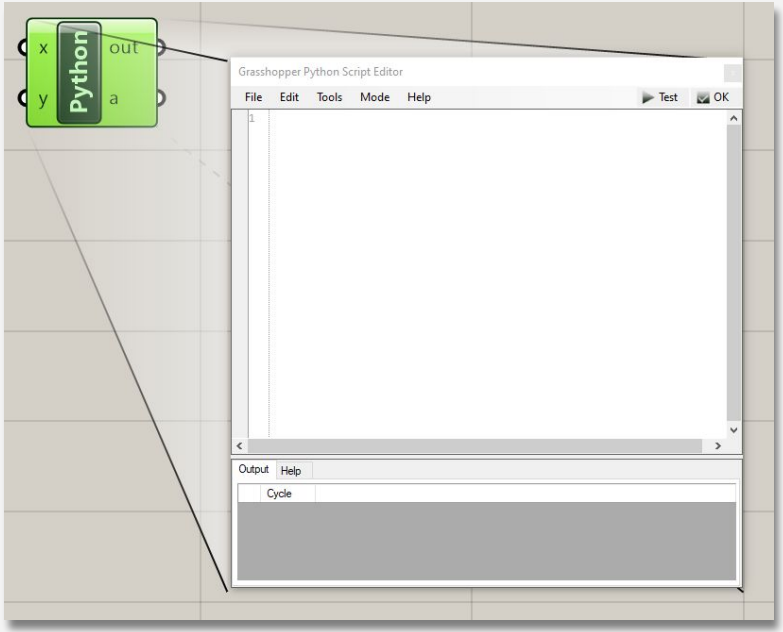
>>> Grasshopper Python

GH Python components anatomy



>>> Input and output parameters

It’s customizable; by zooming in the component you can add, edit, and remove



>>> script editor

To edit the Python script in the component, simply double-click the component.

>>> Executing Python code

hands on experience

```
print ("Hello World")
print (4+5)
```

>>> Printing

Python is case sensitive

```
# anything after '#' will be ignored
# use this feature to annotate/comment your code
```

"While poorly written code is often manageable, a poorly organized ~~gh document~~ code can be absolute hell."
Alexander, Grasshopper forum

>>> Commenting

Use comments to add helpful note about your code to yourself and others.

>>> Basic data types

type()

```
#use '=' to store a variable.
single_quote = 'Single quote allow you to embed "double" quotes in your string.'
double_quote = "Double quote allow you to embed 'single' quotes in your string."
triple_quote = """Triple quotes allows to embed "double quotes" as well as
'single quotes' in your string.
And can also span across multiple lines."""

print (single_quote)
print (double_quote)
print (triple_quote)
```

>>> string

aka. text

```
#Escape characters" you can't use single/double quote in the middle of text
txt = "We are the so-called \"Geek\" from the east."

"""
other escape characters
\'    Single Quote
\\    Backslash
\n    New Line
"""
escape_txt = "Hello\nWorld!"
print (escape_txt)
```

>>> string

Escape characters

```
#Text Format
txt_1 = "My name is {fname}, I'am {age} y.o.".format(fname = "Bahar", age = 21)
txt_2 = "My name is {0}, I'am {1} y.o.".format("Bahar",21)

print (txt_1)
print (txt_2)
```

>>> string

Text format

```
#integer number
a = 5
b = 6
c = a + b #python can do mathematical operations!
print (c) #print result

#float and integer number
a = 5.0
c = a * 3
#be careful with the dynamic-typed and different data types
print ('c='+ str(c))
```

>>> integer & floats

aka. number

```
#try to answer this result
#you can assign multiple variables in one line
var_1, var_2 = 15, 6
task_1 = ((var_1*4)/var_2)-10

#print your result
```

>>> integer & floats

aka. number

```
#boolean values are only 'True' and 'False'
print (2 == 2)
print (2 >= 5)

print ('t' in 'Python')
```

>>> booleans

True or False

```
#indentation is important in Python
x = 5
if (x>6):
    print('x is larger than 6')
elif (x==3):
    print('x is 6')
else:
    print('x is smaller than 6')
```

White space

>>> statements

Conditional statements (white space/indentation matters)

```
#x is the input param in grasshopper.
if x:
    my_str = "Hello World!"
else:
    my_str = "Nothing to say."
```

>>> statements

Conditional statements (white space/indentation matters)

```
#you can create an empty list
numbers = []
#put your data inside square brackets[]
numbers = [1,2,3]
friends = ['Ahmad', 'Hartono', 'Ratna', 'Dewi']
#get first name from the list (index 0)
print (friends[0])
#get last name from the list (index 3)
print (friends[3])
```

>>> list = []

is a collection data (e.g. numbers, strings, boolean, etc) which is ordered and changeable. List is defined using square brackets []

```
#list can have multiple data types
my_list = ['apel',150,'pisang',77,'jeruk',23]
#you can add more data to list
my_list.append(7)
#use extend method to add more elements
my_list.extend([1000, 'anggur'])
print (my_list)
```

>>> list = []

is a collection data (e.g. numbers, strings, boolean, etc) which is ordered and changeable. List is defined using square brackets []

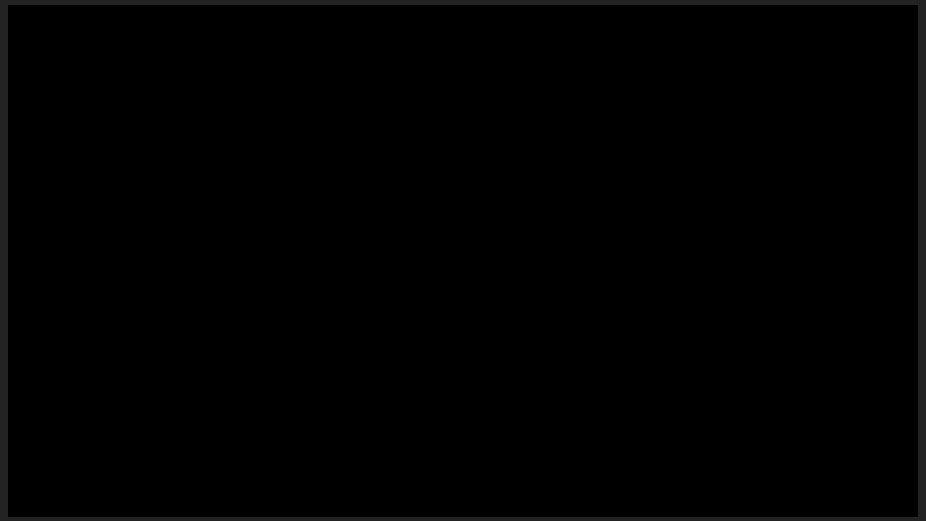
```
#define variable
a = "Hello"
b = "World"
c = a + " " + b

#call method to change the string into lower case
print(c.lower())
#call method to change the string into upper case
print(c.upper())
#get the length of characters
print("jumlah karakter variabel c adalah " + str(len(c)))
```

>>> method

functions(blocks of code to perform an action) that are defined and stored within an object or data type. Check https://docs.python.org/ for full documentation

>>> Break and QnA

Feel free to ask



>>> Python in Rhino (.NET plugins)

RhinoCommon and rhinoscriptsyntax

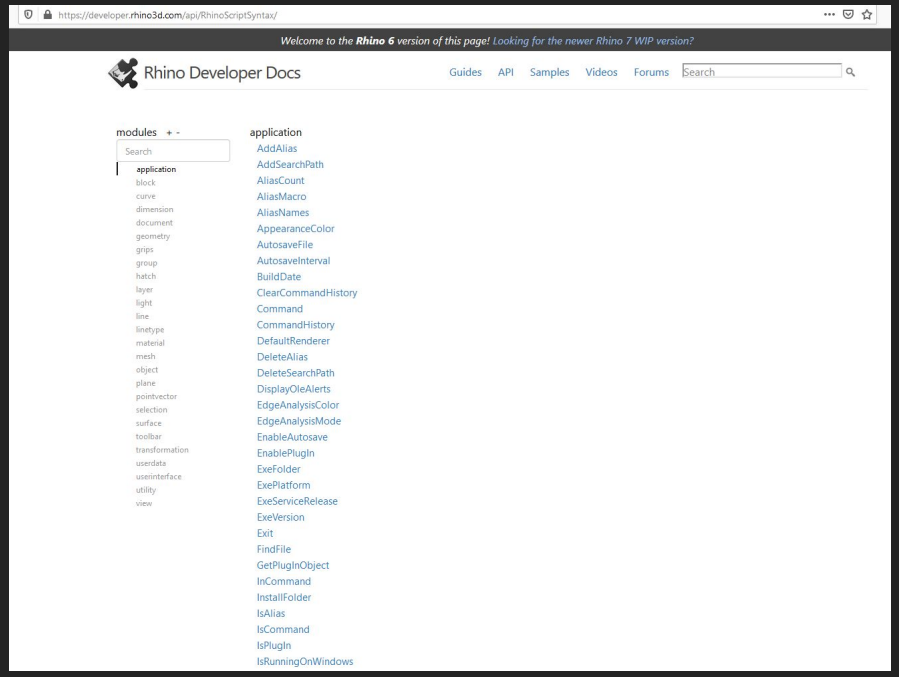
The rhinoscriptsyntax module contains hundreds of easy-to-use functions that perform a variety of operations on Rhino.

The library allows Python to be aware of Rhino's:

- Geometry
- Commands
- Document objects
- Application methods

If you look at the source for the rhinoscriptsyntax functions, they are just python scripts that use RhinoCommon.

>>> Rhinoscriptsyntax



>>> Rhinoscriptsyntax doc

https://developer.rhino3d.com/api/RhinoScriptSyntax/

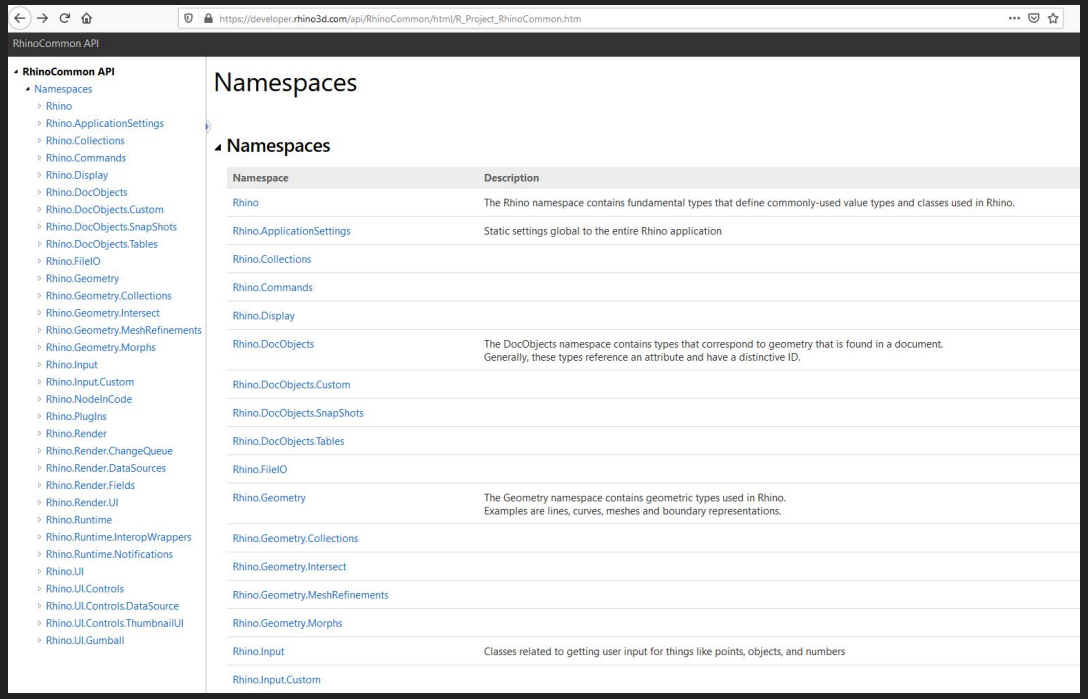
RhinoCommon is the cross-platform .NET plugin SDK (Software Development Kit) available for:

- Rhino for Windows
- Rhino for Mac
- Rhino.Python scripting
- Grasshopper

It allows Python to use all of the classes in the .NET Framework, including the classes available in RhinoCommon.

>>> RhinoCommon

Load additional library into your code.



>>> RhinoCommon

https://developer.rhino3d.com/api/RhinoCommon/

Rhinoscriptsyntax library

```
#create a line with rhinoscriptsyntax
import rhinoscriptsyntax as rs
a = rs.AddLine(x,y)
```

RhinoCommon library

```
#create a line with RhinoCommon and rhinoscriptsyntax
import Rhino.Geometry as rg
import rhinoscriptsyntax as rs
a = rg.Line(rs.coerce3dpoint(x),rs.coerce3dpoint(y))

#create a line with RhinoCommon with type hint
import Rhino.Geometry as rg
a = rg.Line(x,y)
```

>>> imports

Load additional library into your code.

>>> statements_2: looping

an instruction that repeats until a specified condition is reached.

```python
#import RhinoCommon library
import Rhino as rh

#the number of points
x_size = 10
#define a new list
my_points = []

for i in range(x_size): #loop to repeat until max number.
    print(i)
    #use i as 'x' coordinate and store every process to a list
    my_points.append(rh.Geometry.Point3d(i, 0,0))
```

```
import math #mathemathic library
import Rhino as rh #rhinocommon library

#define a new list
math_pts = []

for i in range(500):
    x = math.cos(i*0.1) * i*0.1 #coordinate of x
    y = math.sin(i*0.1) * i*0.08 #coordinate of y
    #conditiona logic of z coordinate
    if(i < 200):
        z = i*0.05
    else:
        z = 20-i*0.05
    #create point w/ x,y,z values
    p = rh.Geometry.Point3d(x,y,z)
    #store the point result into list
    math_pts.append(p)
```

>>> nested loop and nested list

```
#import libraries
import Rhino as rh
import math
import ghpythonlib as ghp

#define a new list
pts = []

for i in range(x):
    for j in range(y):
        #coordinate of z
        pt_z = math.sin(i*u) * math.sin(j*u) * z
        #create point geometry
        pt = rh.Geometry.Point3d(i,j, pt_z)
        #store the point to a list
        pts.append(pt)

#code continues to the next page...
```

```
#define more new list
nested_pts, my_polylines = [], []

i = 0
#this is will loop until reach the length of pts
while i < len(pts):
    nested_pts.append(pts[i:i+y]) #list slicing
    i += y

for i in nested_pts:
    #create a polyline from list of nested points
    my_polyline = rh.Geometry.Polyline(i)
    #convert polyline to a curve, check RhinoCommon doc for detail
    my_polylines.append(my_polyline.ToPolylineCurve())

#convert python nested list into Grasshopper tree structures
nested_pts = ghp.treehelpers.list_to_tree(nested_pts)
```

>>> function

Named blocks of code that perform an action. Has ability to receive arguments and return values.

```python
import Rhino as rh
import math

def spiral(n):
    """use this to describe your function
    including the input parameters,
    and output parameters"""
    #define a new list
    math_pts = []
    for i in range(n):
        x = math.cos(i*0.1) * i*0.1 #x coordinate
        y = math.sin(i*0.1) * i*0.08 #y coordinate
        z = i*0.05 if i<200 else 20-i*0.05   #list comprehension for z coord.
        p = rh.Geometry.Point3d(x,y,z) #create point geometry
        math_pts.append(p) #add point to list
    return math_pts #function results

#this is where Grasshopper execute your function
a = spiral(500)
```

>>> closing & tips

things that you need to remember.

```
###Google Style####
"""Create spiral points based on mathematical equations

Args:
    n (int): number of points in spiral
Returns:
    math_pts (list): a list of spiral points
"""
```

>>> function documentation styles

Option 1: Google styles
(always annotate your code to help you/others to understand the code)

```
###reSt Style###
"""Create spiral points based on mathematical equations

:param n: number of points in spiral
:type n: int

:returns math_pts: a list of spiral points
:rtype math_pts: list
"""
```

>>> function documentation styles

Option 2: reSt styles

```
###NumPy Style###
"""Create spiral points based on mathematical equations

Parameters
----------
n: int
    number of points in spiral

Returns
-------
math_pts: list
    a list of of spiral points
"""
```

>>> function documentation styles

Option 3: NumPy styles

Resources:

- docs.python.org
- ironpython.net
- RhinoCommon api docs
- Rhino discourse forum
- Google search

>>> read & ask the internet

Skim through; revisit later.

>>> leverage communities

Stackoverflow, github, discourse.mcneel.com, etc

- Code frequently and write it out!
- Ask ‘GOOD’ questions:
  - Give context
  - Outline things you have tried to fix the issue,
  - Offer your best guess to the problem
  - Demo what is happening, include the code / screenshot, error message.
- Make something (trial and error is the best teachers)
- Take breaks

>>> make it stick

>>> thank you