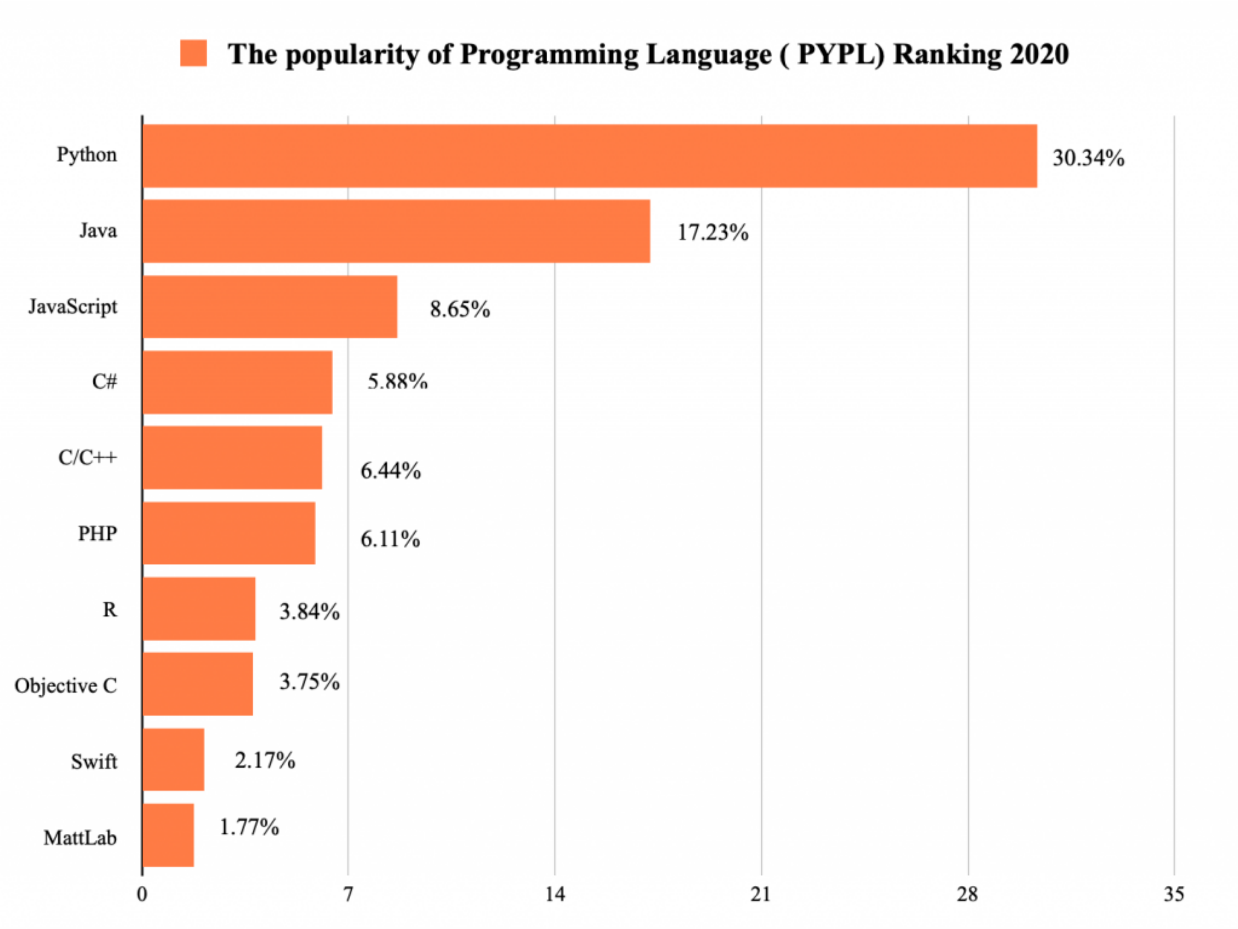
Why python?

Python is easy-to-use, and easy-to-deploy programming language. It provides excellent library support and has a large developer community. The programming language provides a great starting point for beginners. Python is the most popular programming languages in 2020.



Link check in 2021:

- **Check current update**

What is Python used for?

- 1. AI and machine learning
- 1. Data analytics
- 1. Data visualisation
- 1. Programming applications
- 1. Web development
- 1. Game development

What types of jobs use Python?

- Developer
- Data analyst
- Data scientist
- Ethical hacker/penetration tester
- Software engineer
- Data journalist
- Cloud architect
- QA engineer

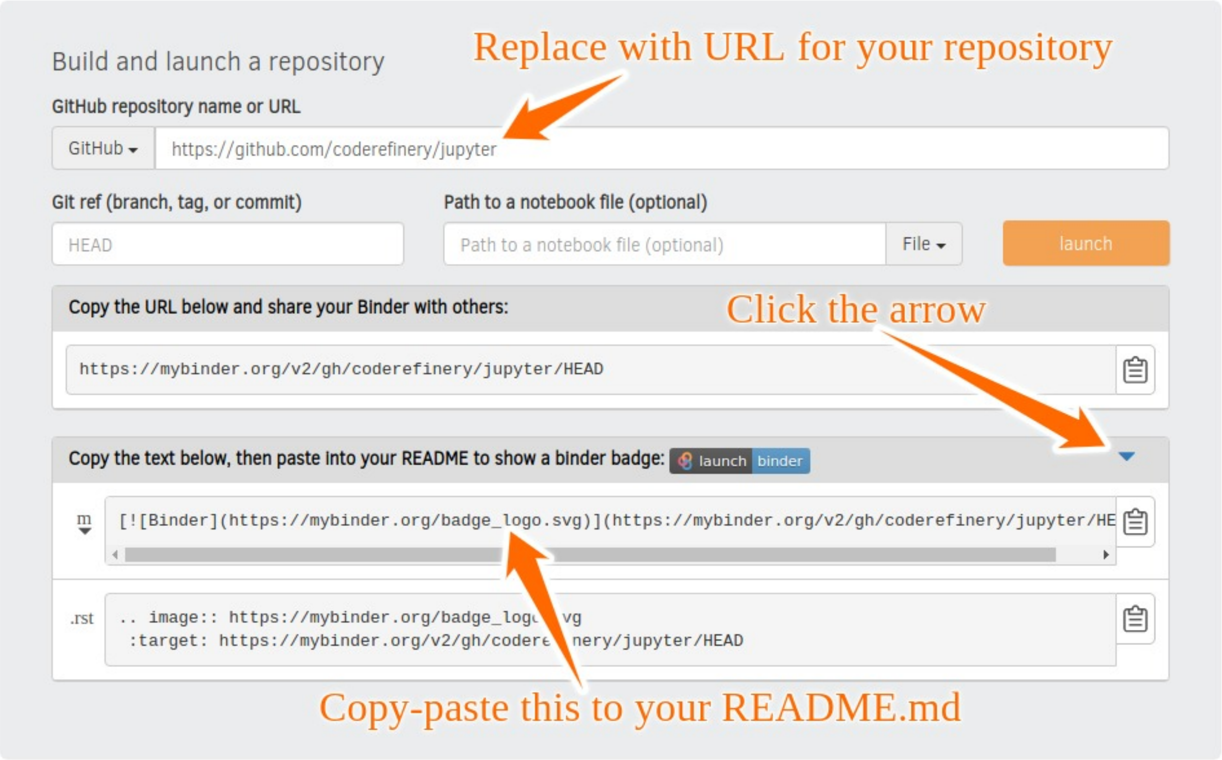
Jupyter Notebook

The Jupyter Notebook

> *The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.*

Jupiter Notebook Example

Go to **mybinder web page** pate the github link https://github.com/jvdkwast/Python3_Jupyter_Notebook into GitHub repository name or URL tab and press launch.



Replace with URL for your repository

Build and launch a repository

GitHub repository name or URL

GitHub ▾ https://github.com/coderefinery/jupyter

Git ref (branch, tag, or commit)

HEAD

Path to a notebook file (optional)

Path to a notebook file (optional) File ▾ launch

Copy the URL below and share your Binder with others:

Click the arrow

```
https://mybinder.org/v2/gh/coderefinery/jupyter/HEAD
```

Copy the text below, then paste into your README to show a binder badge: launch binder

m

```
[![Binder](https://mybinder.org/badge_logo.svg)](https://mybinder.org/v2/gh/coderefinery/jupyter/HE
```

.rst

```
.. image:: https://mybinder.org/badge_logo.svg
 :target: https://mybinder.org/v2/gh/coderefinery/jupyter/HEAD
```

Copy-paste this to your README.md

Or Simple click on the icon below.

launch binder

How easy is python

In [1]:

```
# Check Python version !!
import sys
!{sys.executable} --version
```

```
Python 3.7.1
```

The print Statement

> Whenever we learn a new language, it is an age-old tradition to start by displaying the text "Hello World" on the screen.

In [2]:

```
print("Hello World")
```

```
Hello World
```

The text Hello World is bounded by quotation marks because it is a string or a group of characters, more on this later.

Next, we'll print a few numbers. Each call to print moves the output to a new line:

In [3]:

```
print(50)
print(1000)
print(3.142)
```

```
50
1000
3.142
```

Printing Multiple Pieces of Data

We can even print multiple things in a single print command; we just have to separate them using commas. Let's see this in action:

In [4]:

```
print(50, 1000, 3.142, "Hello World")
```

```
50 1000 3.142 Hello World
```

By default, each print statement prints text in a new line. If we want multiple print statements to print in the same line, we can use the following code:

In [5]:

```
print("Hello", end="")
print("World")

print("Hello", end=" ")
print("World")
```

```
HelloWorld
Hello World
```

Small Exercise:

Use the print function to get the result below:

- The results of a + b: 14
- The results of a^b : 144
- The results of a/b : 6.0

In [6]:

```
# Define values
a = 12
b = 2

# Print the results of a + b
print('The results of a + b: ',a + b)

# Print the results of a^b
print('The results of a^b :',a**b)

# Print the results of a/b
print('The results of a/b :',a/b)
```

```
The results of a + b:  14
The results of a^b : 144
The results of a/b : 6.0
```

Comments

Comments are pieces of text used to describe what is happening in the code. They have no effect on the code whatsoever.

A comment can be written using the # character:

In [7]:

```
print(50)  # This line prints 50
print("Hello World")  # This line prints Hello World

# This is just a comment hanging out on its own!

# For multi-line comments, we must
# add the hashtag symbol
# each time
```

```
50
Hello World
```

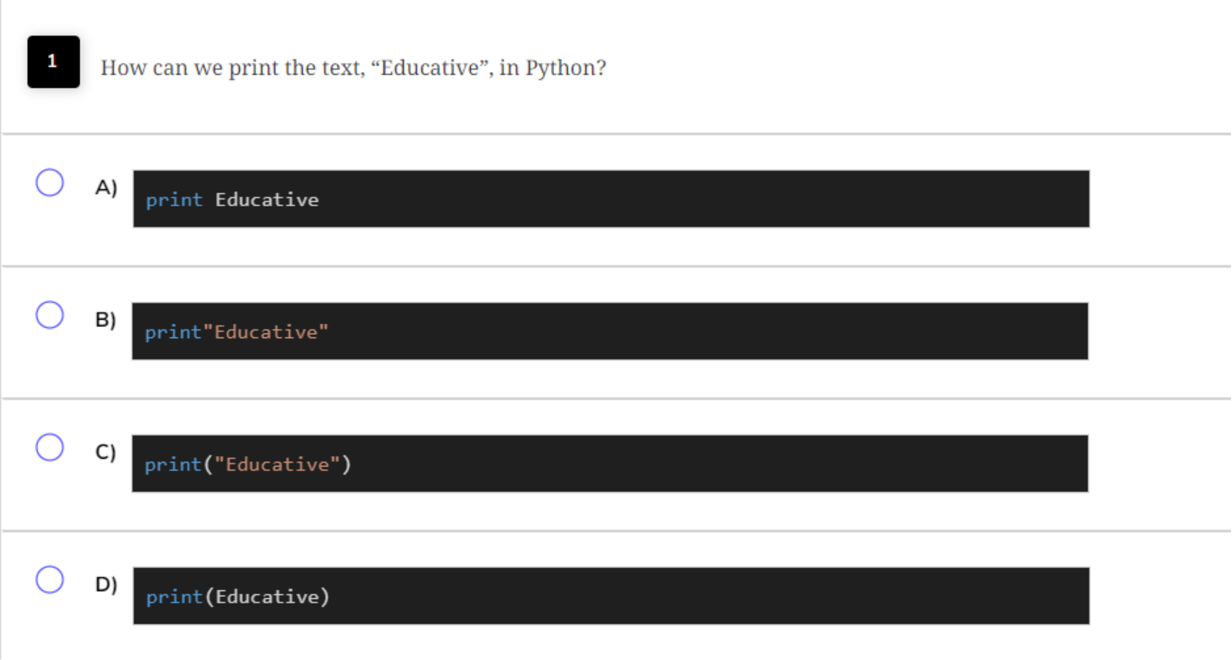
An alternative to these multi-line comments (line 4 - 8) are docstrings. They are encased in triple quotes, """, and can be used to replace multi-line comments:

In [8]:

```
""" Docstrings are pretty cool
for writing longer comments
or notes about the code"""
```

Out[8]:

```
' Docstrings are pretty cool\nfor writing longer comments\nor notes about the code'
```



1 How can we print the text, “Educative”, in Python?

○ A)

```
print Educative
```

○ B)

```
print"Educative"
```

○ C)

```
print("Educative")
```

○ D)

```
print(Educative)
```

Python’s Data Types

> Unlike many other languages, Python does not place a strong emphasis on defining the data type of an object, which makes coding much simpler. The language provides

> three main data types:
>
> - Numbers
> - Strings
> - Booleans

Variables

> A variable is simply a name to which a value can be assigned.

Numbers

Python is one of the most powerful languages when it comes to manipulating numerical data.

It is equipped with support for several types of numbers, along with utilities for performing computations on them.

There are three main types of numbers in Python:

1. ## Integers

The integer data type is comprised of all the positive and negative whole numbers.

The amount of memory an integer occupies depends on its value. For example, 0 will take up 24 bytes whereas 1 would occupy 28 bytes.

Here are some examples of integers:

In [9]:

```
print(10)  # A positive integer
print(-3000)  # A negative integer

num = 123456789  # Assigning an integer to a variable
print(num)
num = -16000  # Assigning a new integer
print(num)
```

```
10
-3000
123456789
-16000
```

Floating Point Numbers

Floating-point numbers, or floats, refer to positive and negative decimal numbers.

Python allows us to create decimals up to a very high decimal place.

This ensures accurate computations for precise values.

A float occupies 24 bytes of memory.

Below, we can find some examples of floats:

In [10]:

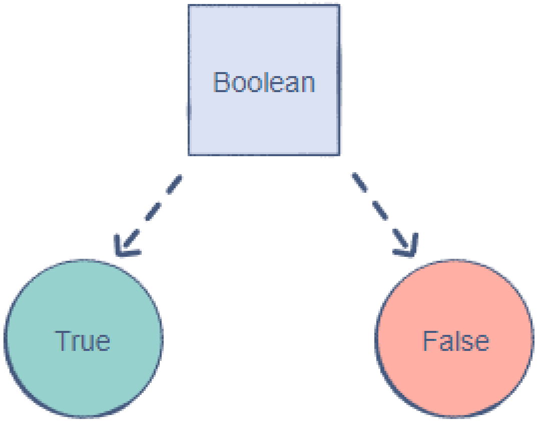
```
print(1.00000000005)  # A positive float
print(-85.6701)  # A negative float

flt_pt = 1.23456789
print(flt_pt)
```

```
1.00000000005
-85.6701
1.23456789
```

Booleans

The Boolean (also known as bool) data type allows us to choose between two values: true and false.



In Python, we can simply use True or False to represent a bool:

In [11]:

```
print(True)

f_bool = False
print(f_bool)
```

```
True
False
```

Strings

A group of characters such as this is an example of the string data type.

A string is a collection of characters closed within single, double or triple quotation marks.

A string can also contain a single character or be entirely empty.

In [12]:

```
print("Harry Potter!")  # Double quotation marks

got = 'Game of Thrones...'  # Single quotation marks
print(got)
print("$")  # Single character

empty = ""
print(empty)  # Just prints an empty line

a = '''Triple quotes allows
multi-line string.'''
print(a)
```

```
Harry Potter!
Game of Thrones...
$

Triple quotes allows
multi-line string.
```

In [13]:

```
print(a)
```

```
Triple quotes allows
multi-line string.
```

The Length of a String

The length of a string can be found using the len() built-in function. This length indicates the number of characters in the string:

In [14]:

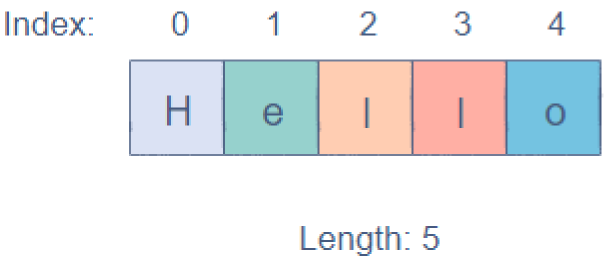
```
random_string = "I am Batman"  # 11 characters
print(len(random_string))
```

```
11
```

Indexing

In a string, every character is given a numerical index based on its position.

A string in Python is indexed from 0 to n-1 where n is its length. This means that the index of the first character in a string is 0.



Accessing Characters

Each character in a string can be accessed using its index. The index must be closed within square brackets, [], and appended to the string.

In [15]:

```
batman = "Bruce Wayne"

first = batman[0]  # Accessing the first character
print(first)

space = batman[5]  # Accessing the empty space in the string
print(space)
```

```
B
```

Exercise: Print the last character

In [16]:

```
#last = batman[len(batman) - 1]
#print(last)
# The following will produce an error since the index is out of bounds
#err = batman[len(batman)]
```

Operator	Purpose	Notation
`()`	Parentheses	Encapsulates the Precedent Operation
`**`	Exponent	In-fix
`%`, `*`, `/`, `//`	Modulo, Multiplication, Division, Floor Division	In-fix
`+`, `-`	Addition, Subtraction	In-fix

# Arithmetic Operators

Operator	Purpose	Notation
`()`	Parentheses	Encapsulates the Precedent Operation
`**`	Exponent	In-fix
`%`, `*`, `/`, `//`	Modulo, Multiplication, Division, Floor Division	In-fix
`+`, `-`	Addition, Subtraction	In-fix

In [17]:

```
print(10 + 5)
```

```
float1 = 13.65
float2 = 3.40
print(float1 + float2)

num = 20
flt = 10.5
print(num + flt)

print(10 - 5)

float1 = -18.678
float2 = 3.55
print(float1 - float2)

num = 20
flt = 10.5
print(num - flt)

print(43 // 10)

float1 = 5.5
float2 = 4.5
print(5.5 // 4.5)
print(12.4 // 2)
```

```
15
17.05
30.5
5
-22.228
9.5
4
1.0
6.0
```

Comparison Operators

Operator	Purpose	Notation
`>`	Greater Than	In-fix
`<`	Less Than	In-fix
`>=`	Greater Than or Equal To	In-fix
`<=`	Less Than or Equal To	In-fix
`==`	Equal To	In-fix
`!=`	Not Equal To	In-fix
`is`	Equal To (Identity)	In-fix
`is not`	Not Equal To (Identity)	In-fix

In [18]:

```python
num1 = 5
num2 = 10
num3 = 10
list1 = [6,7,8]
list2 = [6,7,8]

print(num2 > num1)  # 10 is greater than 5
print(num1 > num2)  # 5 is not greater than 10

print(num2 == num3)  # Both have the same value
print(num3 != num1)  # Both have different values

print(3 + 10 == 5 + 5)  # Both are not equal
print(3 <= 2)  # 3 is not less than or equal to 2

print(num2 is not num3)  # Both have the same object
print(list1 is list2)  # Both have the different objects
```

```
True
False
True
True
False
False
False
False
```

Assignment Operators

Operator	Purpose	Notation
`=`	Assign	In-fix
`+=`	Add and Assign	In-fix
`-=`	Subtract and Assign	In-fix
`*=`	Multiply and Assign	In-fix
`/=`	Divide and Assign	In-fix
`//=`	Divide, Floor, and Assign	In-fix
`**=`	Raise power and Assign	In-fix
`%=`	Take Modulo and Assign	In-fix
`|=`	OR and Assign	In-fix
`&=`	AND and Assign	In-fix
`^=`	XOR and Assign	In-fix
`>>=`	Right-shift and Assign	In-fix
`<<=`	Left-shift and Assign	In-fix

In [19]:

```
year = 2019
print(year)

year = 2020
print(year)

year = year + 1  # Using the existing value to create a new one
print(year)

first = 20
second = first
first = 35  # Updating 'first'

print(first, second)  # 'second' remains unchanged

num = 10
print(num)

num += 5
print(num)

num -= 5
print(num)
```

```
num *= 2
print(num)

num /= 2
print(num)

num **= 2
print(num)

# Try all the others here!
```

```
2019
2020
2021
35 20
10
15
10
20
10.0
100.0
```

Logical Operators

Operator	Purpose	Notation
`and`	AND	In-fix
`or`	OR	In-fix
`not`	NOT	Prefix

In [20]:

```
# OR Expression
my_bool = True or False
print(my_bool)

# AND Expression
my_bool = True and False
print(my_bool)

# NOT expression
my_bool = False
print(not my_bool)
```

```
True
False
True
```

Indexing example and exercise

In [21]:

```
my_string = "This is MY string!"
print(my_string[0:7])  # A step of 1
print(my_string[0:7:2])  # A step of 2
print(my_string[0:7:5])  # A step of 5
```

```
This is
Ti s
Ti
```

In [22]:

```
my_string = "This is MY string!"
print(my_string[:8])  # All the characters before 'M'
print(my_string[8:])  # All the characters starting from 'M'
print(my_string[:])  # The whole string
print(my_string[::-1])  # The whole string in reverse (step is -1)
```

```
This is
MY string!
This is MY string!
!gnirts YM si sihT
```

my_string = "0123456789"

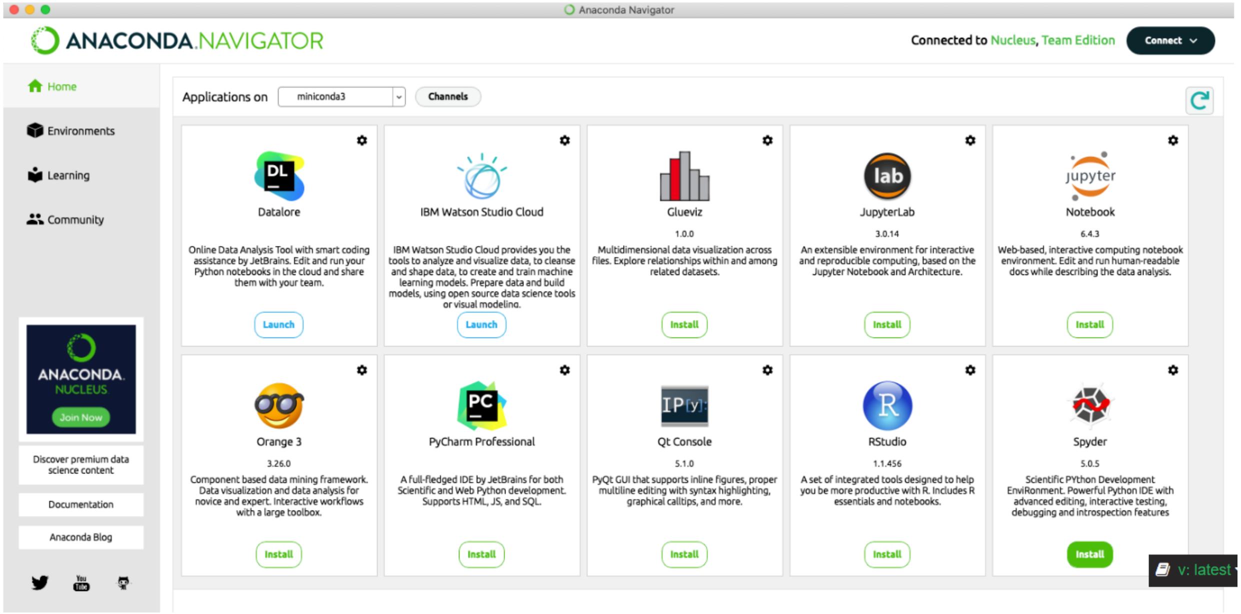
print out: 86 as a results

In [23]:

```
my_string = "0123456789"
# -2 : -6 mean from 8 to 4 in a step of 2
print(my_string[-2: -6: -2])
```

```
86
```

Install Anaconda Navigator



Datalore

Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.

IBM Watson Studio Cloud

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

Glueviz
1.0.0

Multidimensional data visualization across files. Explore relationships within and among related datasets.

JupyterLab
3.0.14

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.4.3

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Orange 3
3.26.0

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

PyCharm Professional

A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.

Qt Console
5.1.0

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

RStudio
1.1.456

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Spyder
5.0.5

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

In [ ]: