CSE3CI:
Computational Intelligence - Lab 1

Objectives: This lab aims to introduce some basics of Python programming language,which has been widely adopted and popular in data sciences.

For students who have some skills and experiences with Python and numpy, you are strongly encouraged to go ahead by yourself; For the newcomers, this lab will serve as a quick access to Python world.

Table of Content

Introduction to Python

Python is a high-level, dynamically typed multi-paradigm programming language. Python code is often said to be almost like pseudocode, since it allows you to express very powerful ideas in very few lines of code while being very readable.

Here is the formal web of Python: https://www.python.org, you can find a lot of useful learning materials here.

Python Versions

There are currently two different supported versions of Python, 2.7.x version, and 3.5.x/3.6.x/3.7.x/3.8.x/3.9.x version. Somewhat confusingly, Python 3.x.x introduced many backward-incompatible changes to the language, so code written for 2.7 may notwork under 3.x and vice versa. For this class, all codes will use Python 3.7 or Python 3.9 on the most popular python platform **Anaconda** for data science. By the way, Python 2 is officially retiring.

Install Anaconda

The reason why we use Anaconda instead of the original Python is because that the open source **Anaconda Distribution** is the easiest way to do Python data science and machine learning.

It includes hundreds of popular data science packages, the conda package and virtual environment manager for Windows, Linux and MacOS. Conda makes it quick and easy to install, run and upgrade complex data science and machine learning environments like scikit-learn, TensorFlow and SciPy. Anaconda Distribution is the foundation of millions of data science projects as well as Amazon Web Services' Machine Learning AMIs and Anaconda for Microsoft on Azure and Windows.

Step 1: Download Anaconda Distribution latest version of Python 3.9 64-Bit version (Windows, MacOS or Linux).

For windows users:

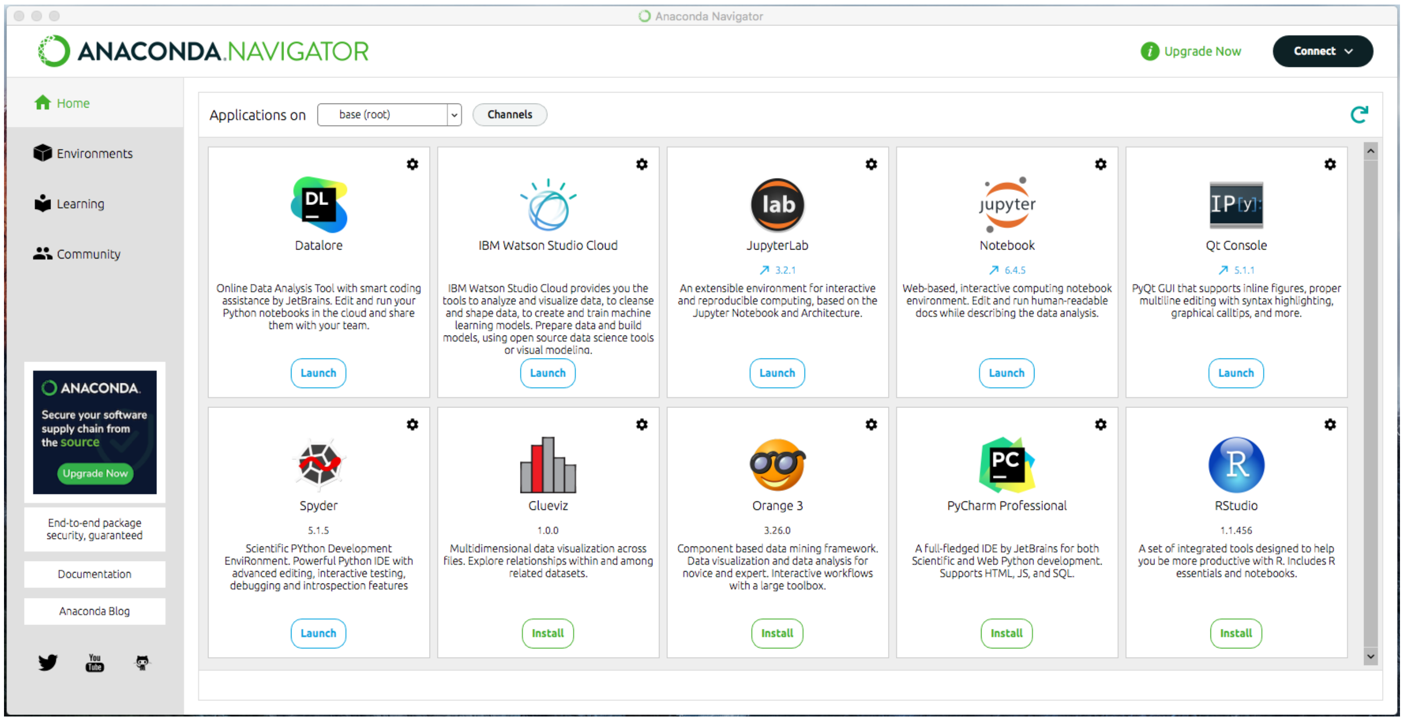
https://repo.anaconda.com/archive/Anaconda3-2021.11-Windows-x86_64.exe

For MacOS users:

https://repo.anaconda.com/archive/Anaconda3-2021.11-MacOSX-x86_64.pkg

Step 2: Install Anaconda with the default settings (unless you know what you are doing.)

Step 3: Open Anaconda Navigator, you will see similar GUI as below, which means you have successfully installed Anaconda, congratulations!



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.
Launch

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.
Launch

JupyterLab
3.2.1
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch

Notebook
6.4.5
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch

Qt Console
5.1.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch

Spyder
5.1.5
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch

Glueviz
1.0.0
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install

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.
Install

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

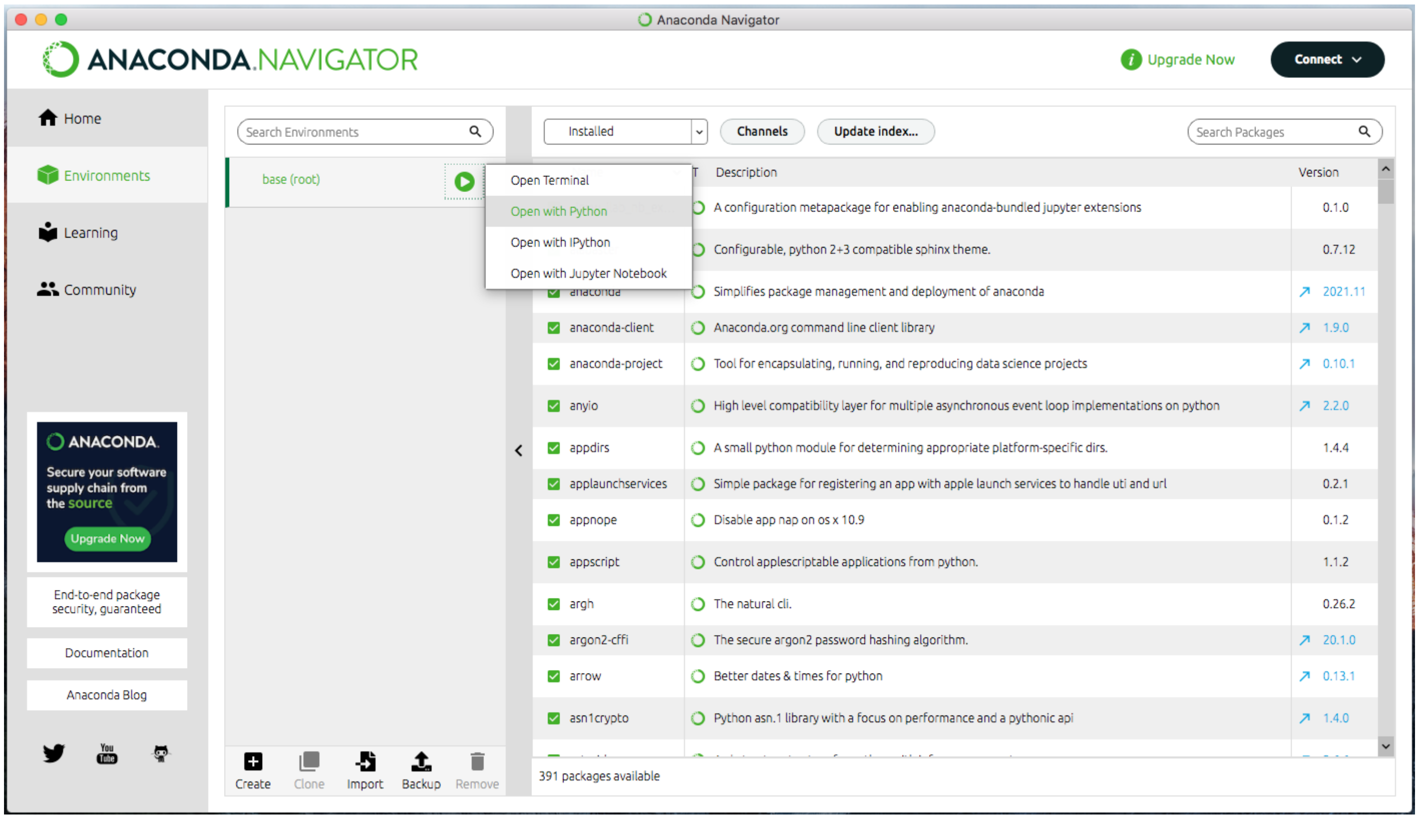
RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.
Install

There are a lot of useful applications on the Home window of Anaconda, we will not cover them all, but we will explain what we need for data analysis in the following courses (Jupyter notebook, Spyder, etc).

For the information, please be noticed that “**Applications on (root)**” means the python developing environment will be used when opening the following applications. The default setting is “root”. We will learn how to create new environment later.

Basic Python

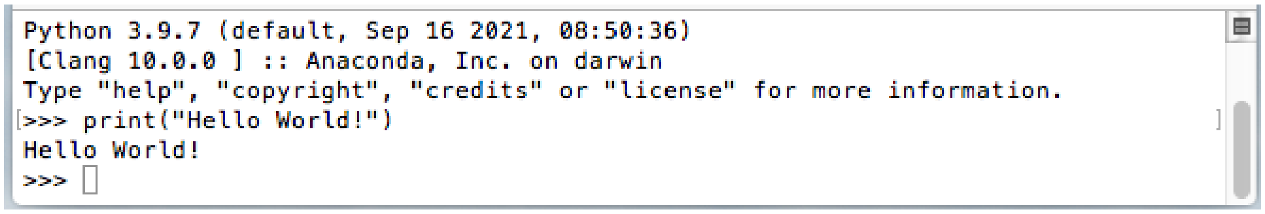
On the Anaconda Navigator, choose **Environments > root (click the Play button) > Open with Python**, you will see a command window of Python popup. You can also find out the version of the Python environment.



	Description	Version
	A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
	Configurable, python 2+3 compatible sphinx theme.	0.7.12
anaconda	Simplifies package management and deployment of anaconda	2021.11
anaconda-client	Anaconda.org command line client library	1.9.0
anaconda-project	Tool for encapsulating, running, and reproducing data science projects	0.10.1
anyio	High level compatibility layer for multiple asynchronous event loop implementations on python	2.2.0
appdirs	A small python module for determining appropriate platform-specific dirs.	1.4.4
applaunchservices	Simple package for registering an app with apple launch services to handle uti and url	0.2.1
appnope	Disable app nap on os x 10.9	0.1.2
appscript	Control applescriptable applications from python.	1.1.2
argh	The natural cli.	0.26.2
argon2-cffi	The secure argon2 password hashing algorithm.	20.1.0
arrow	Better dates & times for python	0.13.1
asn1crypto	Python asn.1 library with a focus on performance and a pythonic api	1.4.0

391 packages available

In the command window, type: **print(“Hello World!”)**, you will see the results as follows.



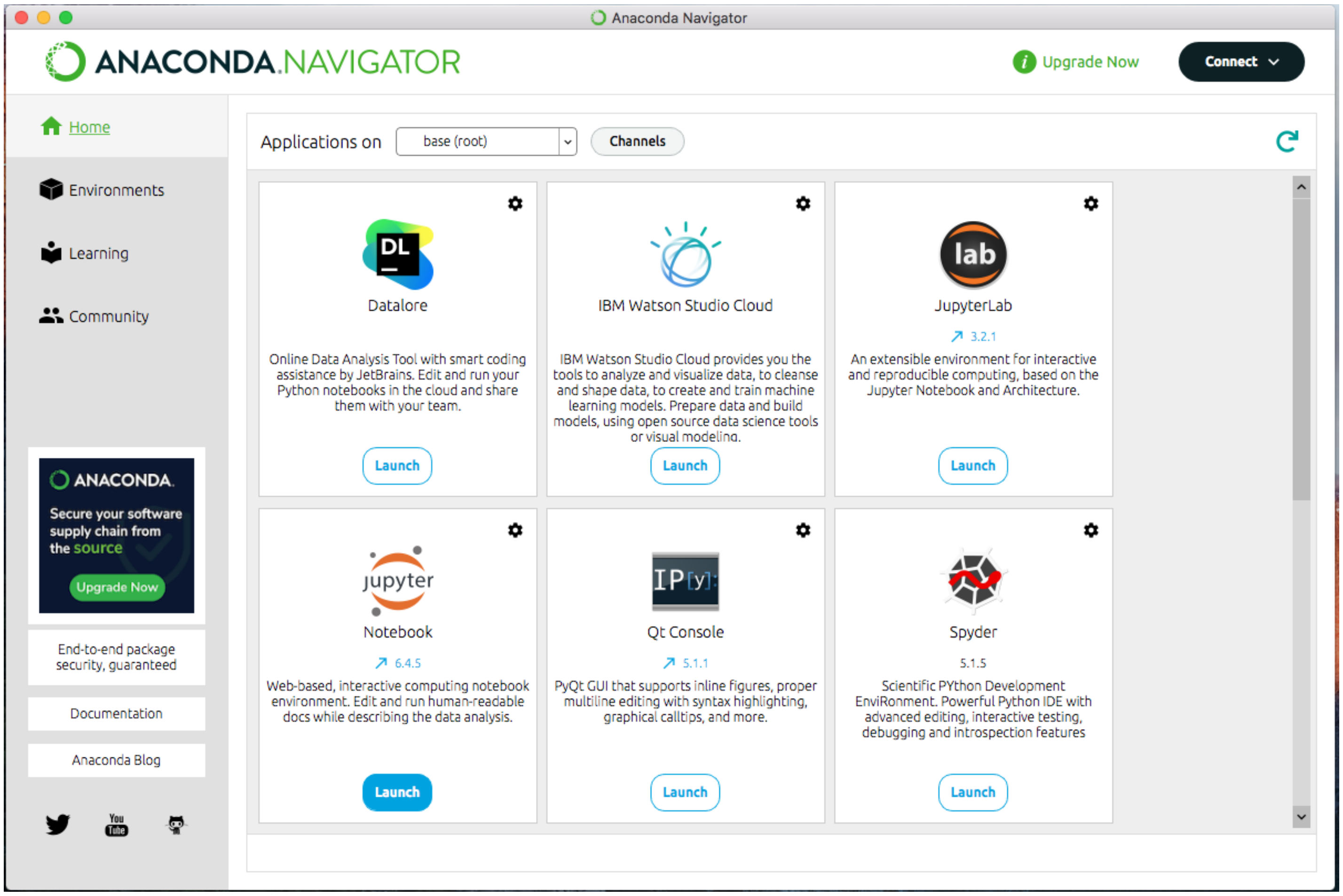
```
Python 3.9.7 (default, Sep 16 2021, 08:50:36)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World!")
Hello World!
>>>
```

Python IDEs for Data Science

The best Python IDEs (Integrated Development Environment) for data science that make data analysis and machine learning easier! It’s a coding tool which allows you to write, test and debug your code in an easier way, as they typically offer code completion or code insight by highlighting, resource management, debugging tools. Even though the IDE is a strictly defined concept, it’s starting to be redefined as other tools such as Jupyter notebook starts gaining more and more features that traditionally belong to IDEs. For example, debugging your code is also possible in **Jupyter Notebook** (aka. IPython) and **Spyder**.

Jupyter Notebook

On the Anaconda Navigator, choose **home > Jupyter Notebook (click the launch button)**, that is highlighted as follows:



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 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.

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

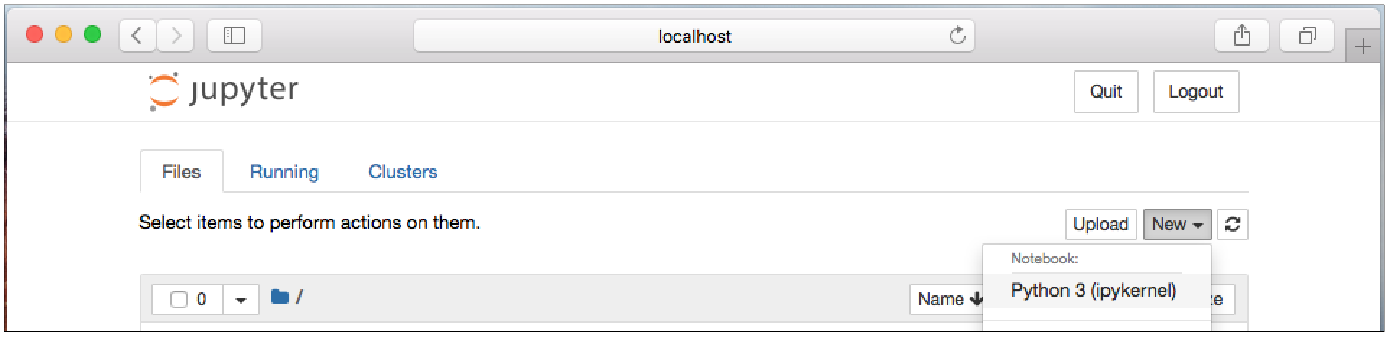
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

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

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

It will start a webpage on your default browser, go to the **Desktop** folder and select **New > Python 3**, it will generate a new notebook with Python 3.

Another way to start Jupyter notebook is **Start > Anaconda3 > Jupyter Notebook.**

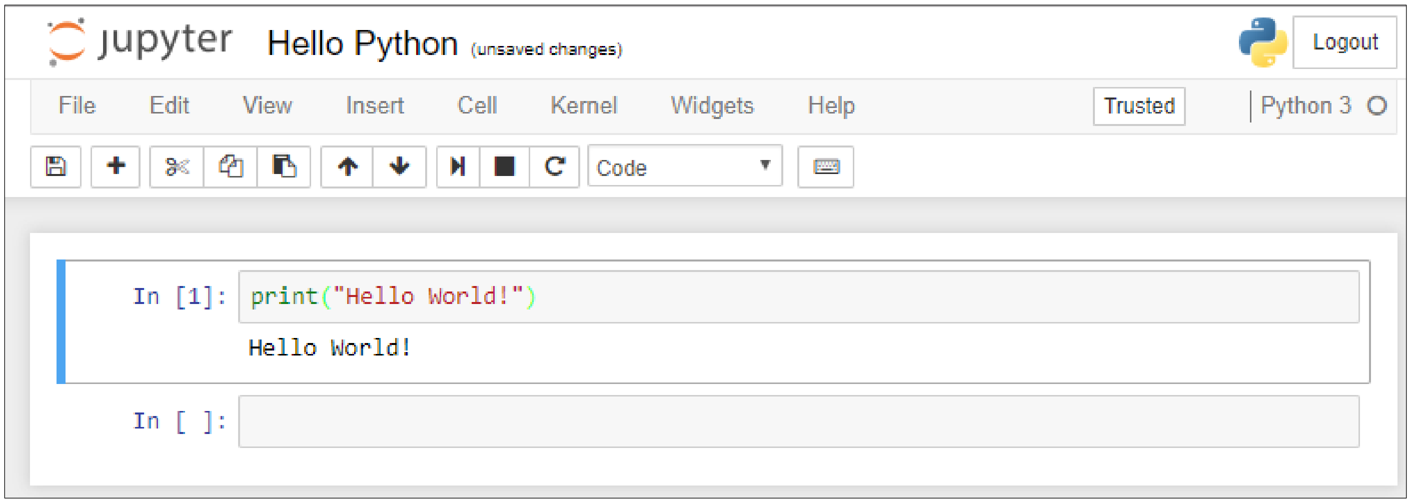


Select items to perform actions on them.

Rename the notebook as "Hello Python" and in the command line (cells) Type: ***print("Hello World!");*** Then select **Cell > Run Cells**, you will get the following results:

Save the file, you will find there is a file named **"Hello Python.ipynb"** on your desktop. You can edit it with the notebook and share it with others if you like.

We will not cover the details of how to use Jupyter notebook, because we assumed the students are smart enough to figure out the details by themselves. Find more information on the documentation.

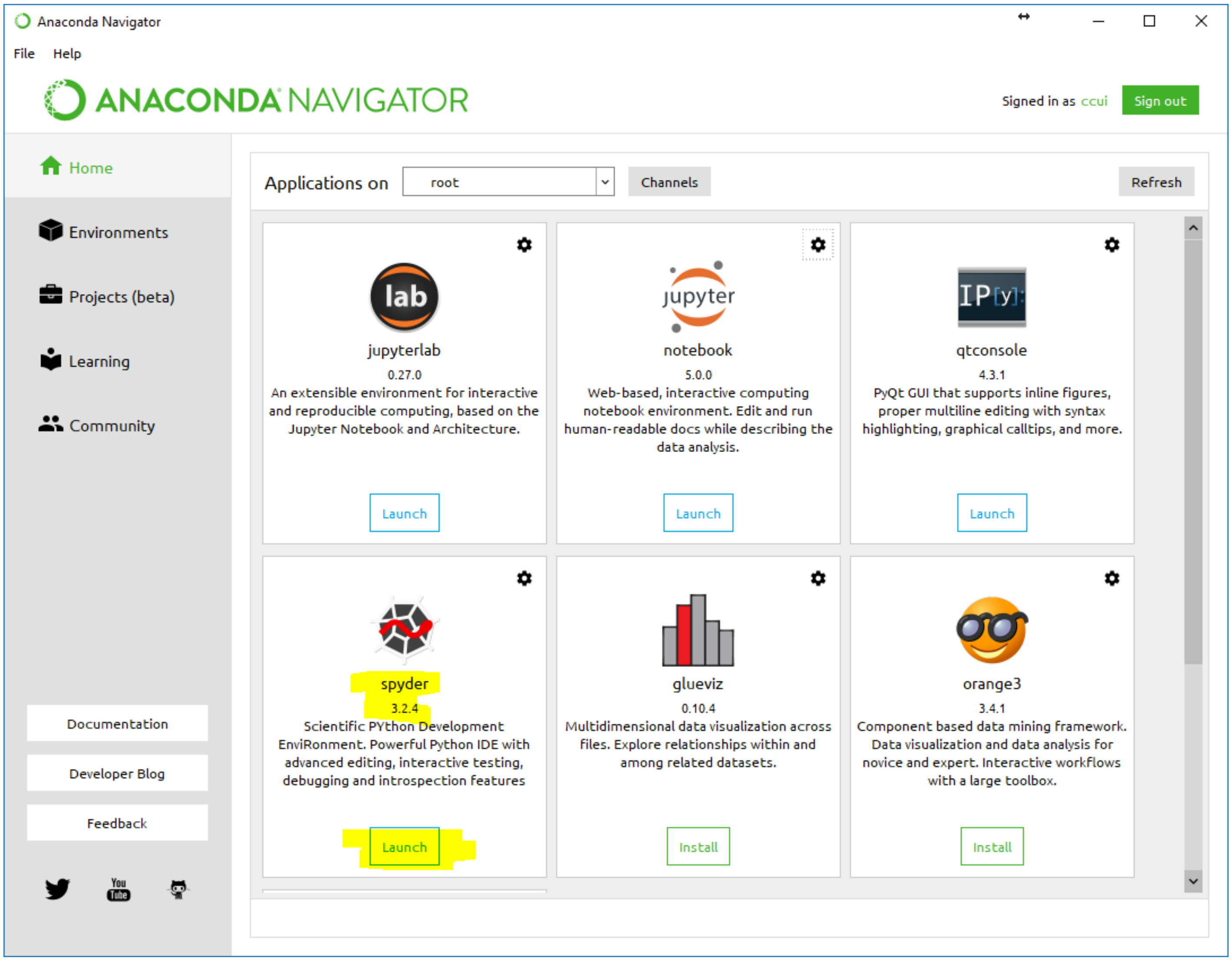


```
In [1]: print("Hello World!")
        Hello World!
In [ ]:
```

Spyder

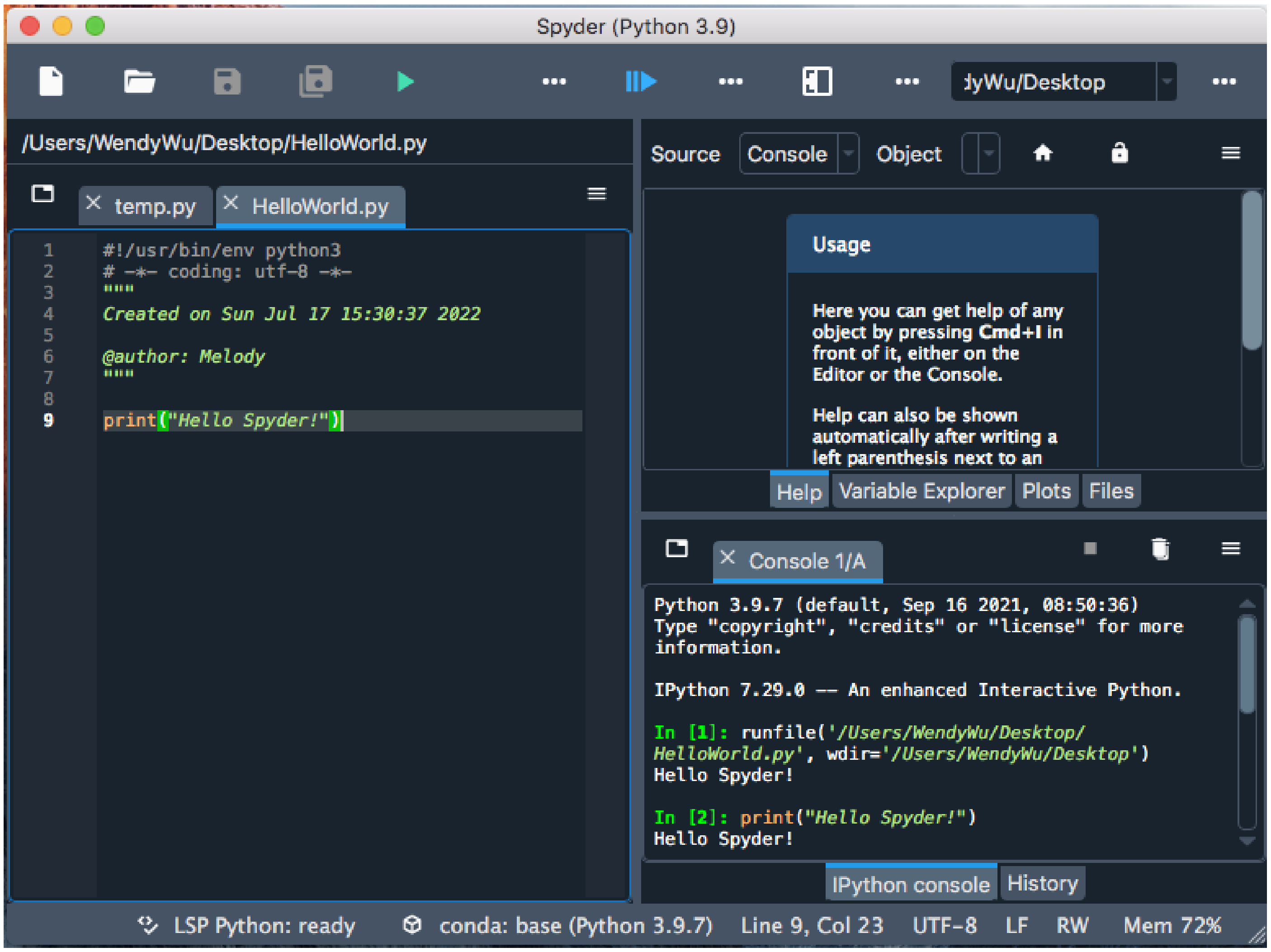
Spyder is an open source cross-platform IDE for data science. If you have never worked with an IDE, Spyder could perfectly be your first approach. If you are switching from **Matlab** or **Rstudio** to Python; Spyder is the way to go, It very intuitive for scientific computing.

On the Anaconda Navigator, choose **home > Spyder (click the launch button)**, that is highlighted as follows:


Anaconda Navigator
File
Help
ANACONDA NAVIGATOR
Signed in as ccui
Sign out
Home
Environments
Projects (beta)
Learning
Community
Documentation
Developer Blog
Feedback
Applications on
root
Channels
Refresh
jupyterlab
0.27.0
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
notebook
5.0.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
qtconsole
4.3.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch
spyder
3.2.4
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch
glueviz
0.10.4
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install
orange3
3.4.1
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.
Install

Another way to start Spyder is **Start > Anaconda3 > Spyder.**

It will start IDE, save the untitled file as "HelloWorld.py" on your desktop, type in "**print("Hello Spyder!")**" then click the Run Button (or press "F5"), you will see the results in the right-bottom **IPython console window**. You could also type the command directly in here.



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Jul 17 15:30:37 2022

@author: Melody
"""

print("Hello Spyder!")
```

Here you can get help of any object by pressing Cmd+I in front of it, either on the Editor or the Console.

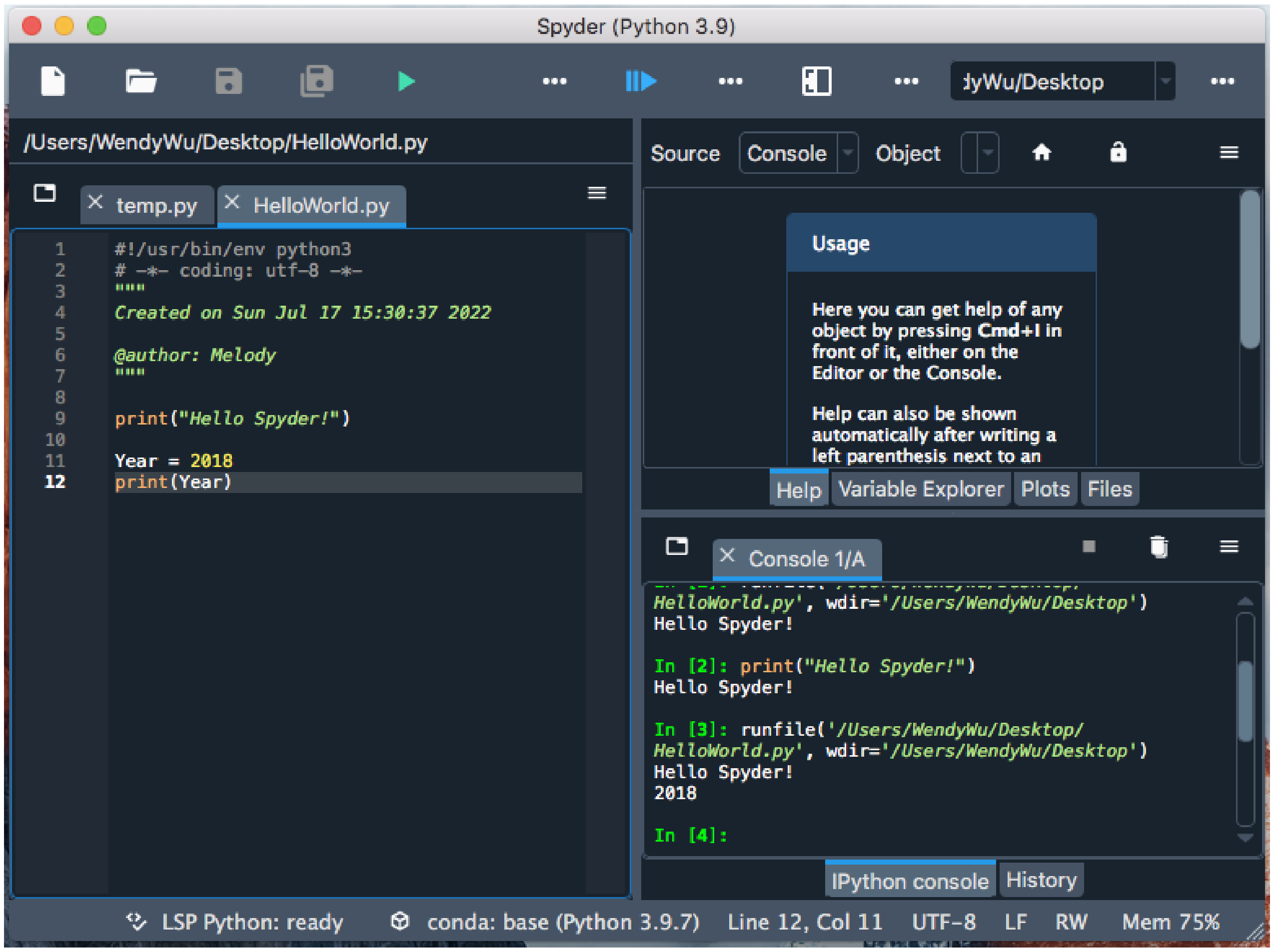
Help can also be shown automatically after writing a left parenthesis next to an

```
Python 3.9.7 (default, Sep 16 2021, 08:50:36)
Type "copyright", "credits" or "license" for more
information.

IPython 7.29.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/WendyWu/Desktop/
HelloWorld.py', wdir='/Users/WendyWu/Desktop')
Hello Spyder!

In [2]: print("Hello Spyder!")
Hello Spyder!
```

Add two lines of code in the HelloWorld.py as follows:

```
Year = 2018
print(Year)
```

Run the code again, you will see the variable “**Year**” appears with more details in the right-top Variable explorer window. This feature is very helpful when we are editing, testing and debugging our codes.



```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Jul 17 15:30:37 2022

@author: Melody
"""

print("Hello Spyder!")

Year = 2018
print(Year)
```

Here you can get help of any object by pressing Cmd+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an

```
HelloWorld.py', wdir='/Users/WendyWu/Desktop')
Hello Spyder!

In [2]: print("Hello Spyder!")
Hello Spyder!

In [3]: runfile('/Users/WendyWu/Desktop/
HelloWorld.py', wdir='/Users/WendyWu/Desktop')
Hello Spyder!
2018

In [4]:
```

Again, we will not explicitly explain how to use the Spyder IDE, we believe the students could learn by themselves. Find more information in the documentation.

The following section has a Jupyter Notebook version and html version available for you to practice.

Programming with Python

Basic data types

Like most languages, Python has many basic types including Integers, Floats, Booleans, and Strings. These data types behave in ways that are like other programming languages (C/C++, Java, MATLAB).

Numbers: Integers and floats work as you would expect from other languages

Note that unlike many languages, Python does **not** have unary increment (***x++***) or decrement (***x--***) operators. “#” starts the comments inline.

```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

Python also has built-in types for complex numbers; you can find all the details in the documentation.

Booleans: Python implements all the usual operators for Boolean logic, but uses English words rather than symbols (&&, ||, etc.):

```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f) # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

Strings: Python has great support for strings:

```
hello = 'hello' # String literals can use single quotes
world = "world" # or double quotes; it does not matter.
print(hello)  # Prints "hello"
print(len(hello))  # String length; prints "5"
hw = hello + ' ' + world # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12) # sprintf style string formatting
print(hw12)  # prints "hello world 12"
```

You can find a list of all string methods in the documentation.

Containers

Python includes several built-in container types: lists, dictionaries, sets, and tuples.

Lists

A list is the Python equivalent of an array, but is resizable and can contain elements of different types:

Note: python starts index from “0”, for example, xs = [3,1,2], this first element of xs is xs[0] that is number 3.

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2]) # Prints "[3, 1, 2] 2"
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
xs[2] = 'foo'     # Lists can contain elements of different types
print(xs)         # Prints "[3, 1, 'foo']"
xs.append('bar') # Add a new element to the end of the list
print(xs)         # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()      # Remove and return the last element of the list
print(x, xs)      # Prints "bar [3, 1, 'foo']"
```

As usual, you can find all the gory details about lists in the documentation.

Slicing: In addition to accessing list elements one at a time, Python provides concise syntax to access sub-lists; this is known as slicing:

```
nums = list(range(5))     # range is a built-in function that creates a list of integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])          # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])           # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])           # Get a slice from the start to index 2 (exclusive); prints "[0,
1]"
print(nums[:])            # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])          # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]        # Assign a new sublist to a slice
print(nums)               # Prints "[0, 1, 8, 9, 4]"
```

We will see slicing again in the context of numpy arrays from **Numpy** Packages later.

Loops: You can loop over the elements of a list like this:

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
# Prints "cat", "dog", "monkey", each on its own line.
```

If you want access to the index of each element within the body of a loop, use the built-in ***enumerate*** function:

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: cat", "#2: dog", "#3: monkey", each on its own line
```

List comprehensions: When programming, frequently we want to transform one type of data into another. As a simple example, consider the following code that computes square numbers:

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)   # Prints [0, 1, 4, 9, 16]
```

You can make this code simpler using a **list comprehension**:

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Prints [0, 1, 4, 9, 16]
```

List comprehensions can also contain conditions:

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

Dictionaries

A dictionary stores (key, value) pairs, like a ***Map*** in **Java** or an **object** in ***Javascript***. You can use it like this:

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data
print(d['cat'])       # Get an entry from a dictionary; prints "cute"
print('cat' in d)     # Check if a dictionary has a given key; prints "True"
d['fish'] = 'wet'     # Set an entry in a dictionary
print(d['fish'])      # Prints "wet"
# print(d['monkey']) # KeyError: 'monkey' not a key of d
print(d.get('monkey', 'N/A')) # Get an element with a default; prints "N/A"
print(d.get('fish', 'N/A'))    # Get an element with a default; prints "wet"
del d['fish']         # Remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

You can find all you need to know about dictionaries in the documentation.

Loops: It is easy to iterate over the keys in a dictionary:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

If you want access to keys and their corresponding values, use the items method:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

Dictionary comprehensions: These are similar to list comprehensions, but allow you to easily construct dictionaries. For example:

```
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)  # Prints "{0: 0, 2: 4, 4: 16}"
```

Sets

A set is an unordered collection of distinct elements. As a simple example, consider the following:

```
animals = {'cat', 'dog'}
print('cat' in animals)   # Check if an element is in a set; prints "True"
print('fish' in animals) # prints "False"
animals.add('fish')       # Add an element to a set
print('fish' in animals)  # Prints "True"
print(len(animals))       # Number of elements in a set; prints "3"
animals.add('cat')        # Adding an element that is already in the set does nothing
print(len(animals))       # Prints "3"
animals.remove('cat')     # Remove an element from a set
print(len(animals))       # Prints "2"
```

As usual, everything you want to know about sets can be found in the documentation.

Loops: Iterating over a set has the same syntax as iterating over a list; however, since sets are unordered, you cannot make assumptions about the order in which you visit the elements of the set:

```
animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: fish", "#2: dog", "#3: cat"
```

Set comprehensions: Like lists and dictionaries, we can easily construct sets using set comprehensions:

```
from math import sqrt
nums = {int(sqrt(x)) for x in range(30)}
print(nums)  # Prints "{0, 1, 2, 3, 4, 5}"
```

Tuples

A tuple is an (immutable) ordered list of values. A tuple is in many ways similar to a list; one of the most important differences is that tuples can be used as keys in dictionaries and as elements of sets, while lists cannot. Here is a trivial example:

```
d = {(x, x + 1): x for x in range(10)}  # Create a dictionary with tuple keys
t = (5, 6)        # Create a tuple
print(type(t))    # Prints "<class 'tuple'>"
print(d[t])       # Prints "5"
print(d[(1, 2)])  # Prints "1"
```

The documentation has more information about tuples.

Functions

Python functions are defined using the def keyword. For example:

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(sign(x))
# Prints "negative", "zero", "positive"
```

We will often define functions to take optional keyword arguments, like this:

```
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True)  # Prints "HELLO, FRED!"
```

There is a lot more information about Python functions in the documentation.

Classes

The syntax for defining classes in Python is straightforward:

```
class Greeter(object):

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance method
    def greet(self, loud=False):
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)

g = Greeter('Fred') # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

You can read a lot more about Python classes in the documentation.

Basic Anaconda

One good feature about Anaconda is its virtual environment manager, it avoids the confusion of different version python interpreters mess your computer default or previous projects. In the follow section, we will learn the basic operations about using environments.

There are two ways to start the Conda Command Line Window:

1. On Windows, go to **Start > Anaconda3 > Anaconda Prompt.**
2. On the Anaconda Navigator, go to **Environments > root > Open Terminal.**

Conda Basics (Optional)

In the Conda Command Window, type the following commands to get familiar with the Anaconda.

Verify conda is installed, check version number:

```
conda info
```

Update conda to the current version:

```
conda update conda
```

Command line help:

```
conda install --help
```

Using Environments

Python is still a developing programming language with an active community. The downside of this is that some predeveloped packages may not work in the newer python version. Yes, it is not always a good choice to use the most advantage technology in real project.

As we know the current python version in Environments>Anaconda3 is version 3.8.11. Assume that you join a team where everyone is working on a project in Python 3.7, we need to create an independent developing environment to work with the team members without intervened by any issues caused by the Python version.

The following steps will help you to install Python 3.7 in an Anaconda environment.

There are two ways to set up the environment of Python 3.7, one is in the Conda Command Window and the other one is on the Anaconda Navigator.

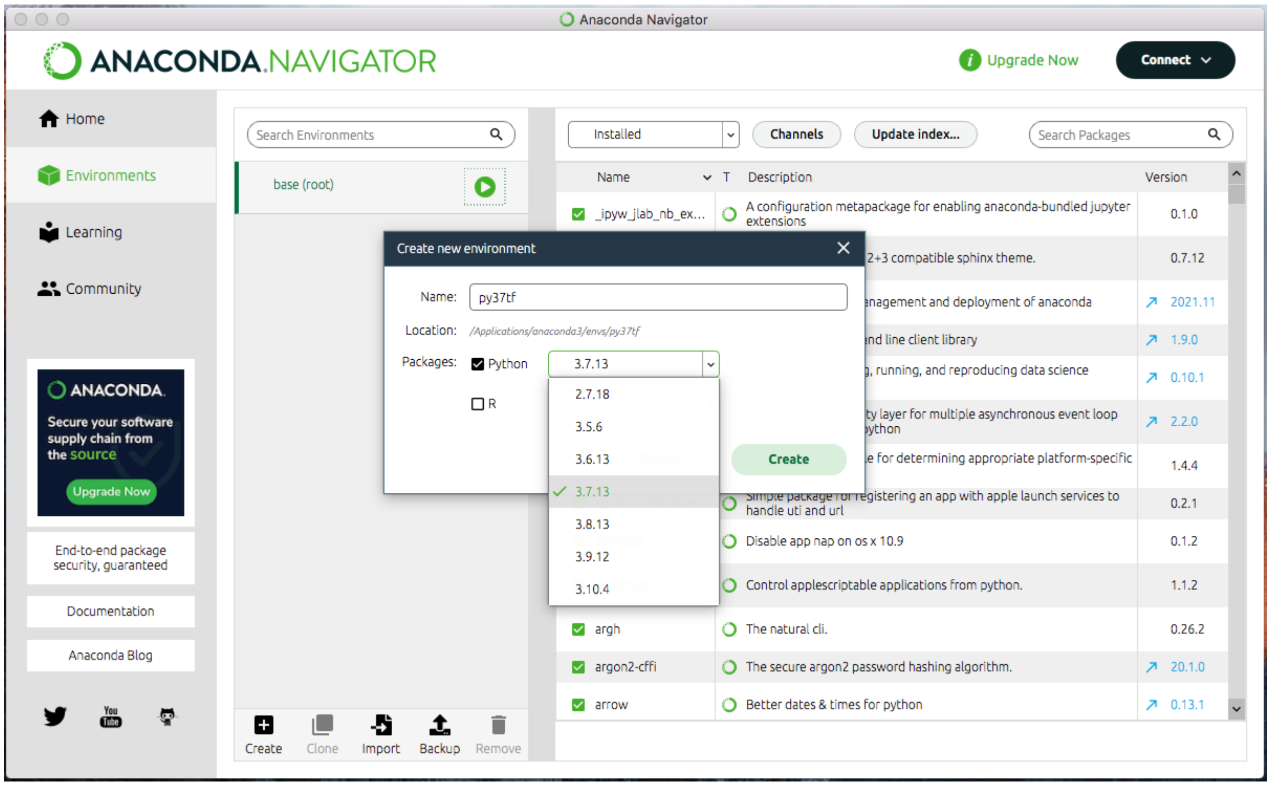
Anaconda GUI

One easy method to create an environment is using the Anaconda Navigator GUI. Launch the Anaconda Navigator from **Start > Anaconda3(64bit) > Anaconda Navigator**. Select the environment.

Using GUI to create an independent python environment is easy. At the bottom of the environment, we can see there are four buttons: **Create, Clone, Import** and **Remove** whose functions are straightforward.

You can now create an environment and name it **"py37tf"** with **Python 3.7.** This part isjust a practice for now, later we will learn how to use Google's TensorFlow framework to build neural network models in this environment.(Note: We use python 3.7 because the tensorflow version 1.15 that we will use is stable for python 3.7).

Click **Create**, in the prop-up window type Name: **py37tf**, in the Packages list:choose **Python 3.7**, then click Create.



Create new environment

Name: py37tf

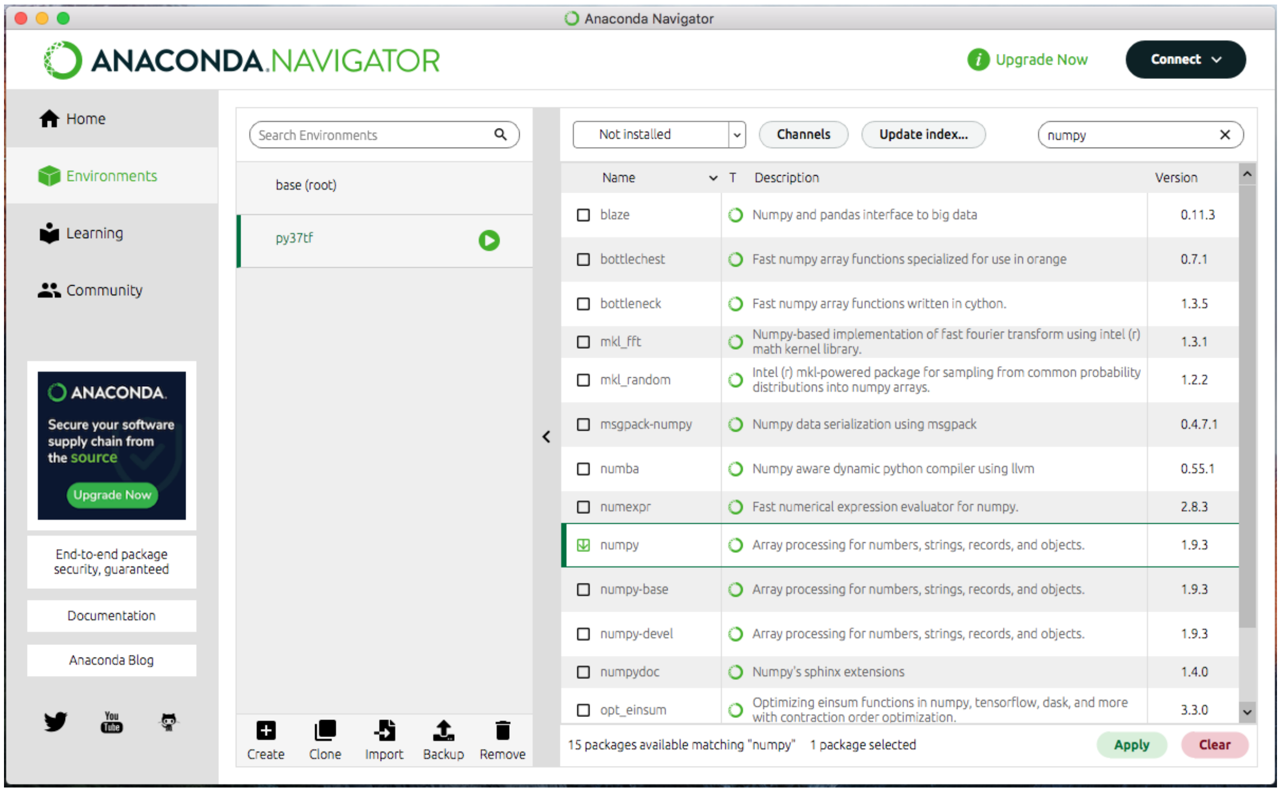
Location: /Applications/anaconda3/envs/py37tf

Packages: [✓] Python 3.7.13 ; [ ] R

2.7.18
3.5.6
3.6.13
✓ 3.7.13
3.8.13
3.9.12
3.10.4

Name	T	Description	Version
✓ _ipyw_jlab_nb_ex...		A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
		2+3 compatible sphinx theme.	0.7.12
		nagement and deployment of anaconda	2021.11
		nd line client library	1.9.0
		, running, and reproducing data science	0.10.1
		ty layer for multiple asynchronous event loop ython	2.2.0
		e for determining appropriate platform-specific	1.4.4
		Simple package for registering an app with apple launch services to handle uti and url	0.2.1
		Disable app nap on os x 10.9	0.1.2
		Control applescriptable applications from python.	1.1.2
✓ argh		The natural cli.	0.26.2
✓ argon2-cffi		The secure argon2 password hashing algorithm.	20.1.0
✓ arrow		Better dates & times for python	0.13.1

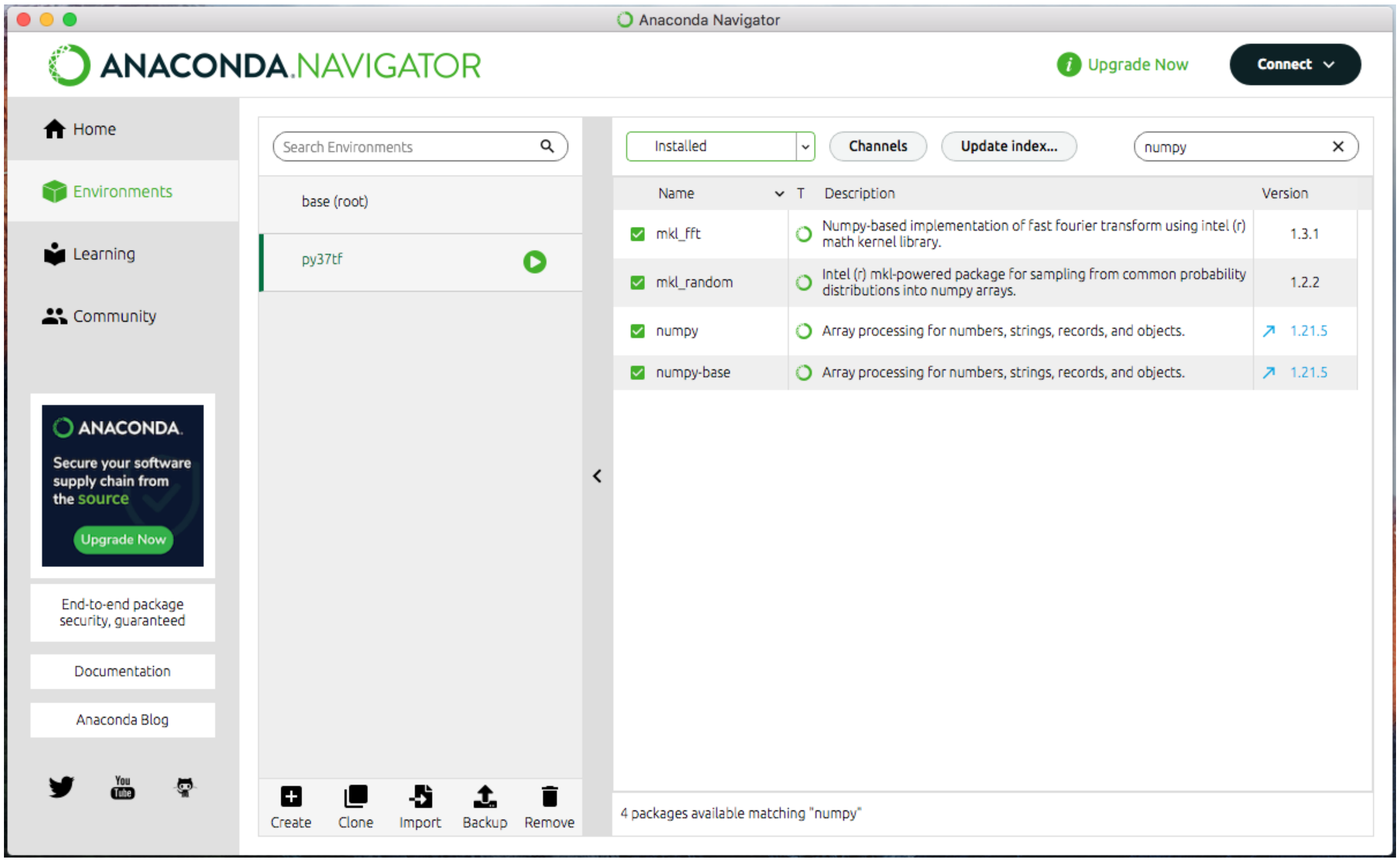
Then, let's install **numpy** in this py37tf environment, change the package into"**Not installed**" and type: **numpy** in the Search Package bar Tick the **numpy** packagein the Name list, and then click **Apply**.



Name	T	Description	Version
☐ blaze		Numpy and pandas interface to big data	0.11.3
☐ bottlechest		Fast numpy array functions specialized for use in orange	0.7.1
☐ bottleneck		Fast numpy array functions written in cython.	1.3.5
☐ mkl_fft		Numpy-based implementation of fast fourier transform using intel (r) math kernel library.	1.3.1
☐ mkl_random		Intel (r) mkl-powered package for sampling from common probability distributions into numpy arrays.	1.2.2
☐ msgpack-numpy		Numpy data serialization using msgpack	0.4.7.1
☐ numba		Numpy aware dynamic python compiler using llvm	0.55.1
☐ numexpr		Fast numerical expression evaluator for numpy.	2.8.3
☑ numpy		Array processing for numbers, strings, records, and objects.	1.9.3
☐ numpy-base		Array processing for numbers, strings, records, and objects.	1.9.3
☐ numpy-devel		Array processing for numbers, strings, records, and objects.	1.9.3
☐ numpydoc		Numpy's sphinx extensions	1.4.0
☐ opt_einsum		Optimizing einsum functions in numpy, tensorflow, dask, and more with contraction order optimization.	3.3.0

15 packages available matching "numpy"   1 package selected

Go back to the Installed package list, you will see the **numpy** is installed. We could use the similar operation to delete a package.



Name	T	Description	Version
✓ mkl_fft		Numpy-based implementation of fast fourier transform using intel (r) math kernel library.	1.3.1
✓ mkl_random		Intel (r) mkl-powered package for sampling from common probability distributions into numpy arrays.	1.2.2
✓ numpy		Array processing for numbers, strings, records, and objects.	1.21.5
✓ numpy-base		Array processing for numbers, strings, records, and objects.	1.21.5

4 packages available matching "numpy"

Click the Play button on the "**py37tf > Open with Python**" to open a python terminal and type the following code to test the installation:

```
import numpy as np
print(np.version.version)
# 1.21.5
```

To delete an environment is just select the one in the list and click the **Remove** button. By the way, you may notice that you cannot remove the **root** environment.

In our next lab, we will start to learn how to use Numpy package in python for data science.

Conda Command Window

Another way to create a new environment, is through the Conda Command Window. In the Conda command window you can create a new environment named "***py37tf***", install Python3.7, by invoking the following command::

```
conda create --name py37 python=3.7
```

The anaconda will suggest that several packages will by installed, type “**y**” and wait the installation done.

Activate the conda environment by issuing the following command:

```
activate py37
```

*LINUX, macOS: source activate py37

Then, you can check your python version by type:

```
 python --version
```

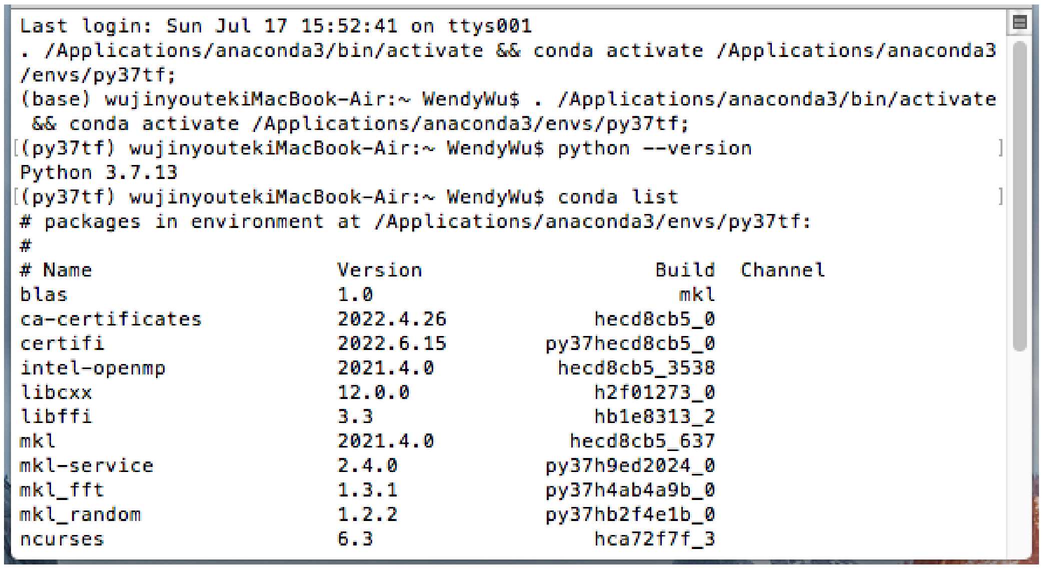
You will get the results like: **Python 3.7.10: Anaconda, Inc.** This means that this environment is Python 3.7.

Installing and updating packages

Now, we fist need to list all packages and versions installed in active environment, enter the following command:

```
conda list
```

Then in the command window, you should be able to find as follows:



```
Last login: Sun Jul 17 15:52:41 on ttys001
. /Applications/anaconda3/bin/activate && conda activate /Applications/anaconda3
/envs/py37tf;
(base) wujinyoutekiMacBook-Air:~ WendyWu$ . /Applications/anaconda3/bin/activate
 && conda activate /Applications/anaconda3/envs/py37tf;
(py37tf) wujinyoutekiMacBook-Air:~ WendyWu$ python --version
Python 3.7.13
(py37tf) wujinyoutekiMacBook-Air:~ WendyWu$ conda list
# packages in environment at /Applications/anaconda3/envs/py37tf:
#
# Name                    Version                   Build  Channel
blas                      1.0                         mkl
ca-certificates           2022.4.26            hecd8cb5_0
certifi                   2022.6.15        py37hecd8cb5_0
intel-openmp              2021.4.0          hecd8cb5_3538
libcxx                    12.0.0               h2f01273_0
libffi                    3.3                  hb1e8313_2
mkl                       2021.4.0           hecd8cb5_637
mkl-service               2.4.0            py37h9ed2024_0
mkl_fft                   1.3.1            py37h4ab4a9b_0
mkl_random                1.2.2            py37hb2f4e1b_0
ncurses                   6.3                  hca72f7f_3
```

Assume we want to install a new package in the active environment (py37t), enter the command:

```
conda install numpy
```

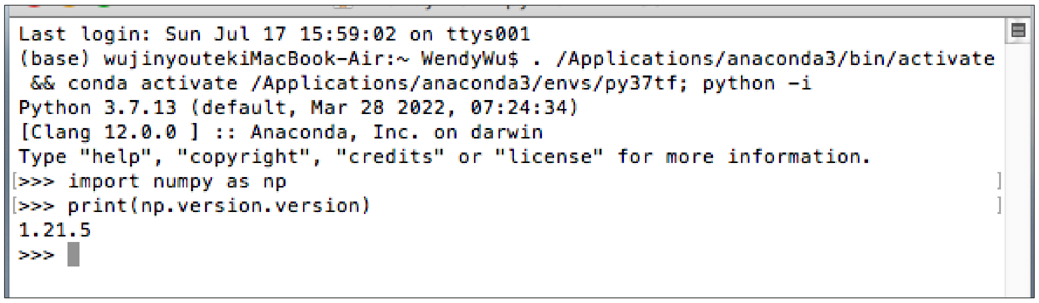
and then enter **"y"** to proceed.

Check the package Numpy is properly installed by running ***"conda list"*** again.

Or you can try the following code to use the Numpy package.

```
python
import numpy as np
print(np.version.version)
quit()
```

You will see the similar results as followings.



```
Last login: Sun Jul 17 15:59:02 on ttys001
(base) wujinyoutekiMacBook-Air:~ WendyWu$ . /Applications/anaconda3/bin/activate
 && conda activate /Applications/anaconda3/envs/py37tf; python -i
Python 3.7.13 (default, Mar 28 2022, 07:24:34)
[Clang 12.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> print(np.version.version)
1.21.5
>>>
```

i

Update a package in the current environment, for example

```
conda update numpy
```

Deactivate the current environment:

```
deactivate
```

*LINUX, macOS: source deactivate

You can find more conda commands from this documentation.