Introduction to Python 3
numpy / keras / matplotlib / scikit-learn

Dr. Ioannis Chamodrakas
Instructor (E.DI.P.)

Python 3

- A general purpose programming language
- Compiled to Python bytecode and executed by Python VM
- Interactive prompt is often called the "Python interpreter"
- Powerful environment for scientific computing and machine learning through libraries:
  - numpy, scipy, matplotlib, scikit-learn

Outline

- **Python 3.7 installation / program execution**
- **Basic Python**
  - Basic data types / Containers (lists, dictionaries, sets, tuples) / Functions / Classes
- **numpy**
  - Arrays, array indexing, datatypes, array math, broadcasting
- **Keras (TensorFlow API)**
  - Installation, NN Build, Compile, Train, Evaluate and Predict, Load & Process a saved model
- **matplotlib**
  - Plotting, subplots, images
- **scikit-learn**
  - nearest neighbours, k-means clustering

Python 3.7 installation

- **Linux Ubuntu / Debian / LinuxMint distributions**
- **Prerequisites**
  - $ sudo apt-get install build-essential checkinstall
  - $ sudo apt-get install libreadline-gplv2-dev libncursesw5-dev \
    libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev libffi-dev
- **Download Python**
  - $ cd /usr/src
  - $ sudo wget https://www.python.org/ftp/python/3.7.2/Python-3.7.2.tgz
  - $ sudo tar xzf Python-3.7.2.tgz
- **Compile Python source**
  - $ cd Python-3.7.2 && sudo ./configure --enable-optimizations
  - $ sudo make altinstall
- **Check Python version**
  - $ python3.7 –V

Installation (continued)

- `$ pip3.7 install --user numpy`
- `$ pip3.7 install --user scipy`
- `$ pip3.7 install --user scikit-learn`
- `$ pip3.7 install --user ipython`

Program execution

- **Interactive prompt**
  - $ python3.7 (CTRL-D or quit() to exit)
- **Running a program from shell**
  - $ python3.7 filename.py
- **Running a program using matplotlib**
  - $ipython filename.py

Code sample

```
x = 34 - 23                       # A comment.
y = "Hello"                       # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
     x = x + 1
     y = y + " World"             # String concat.
print(x)
print(y)
```

Whitespace

- **Whitespace is meaningful in Python: especially indentation and placement of newlines.**
- **Use a newline to end a line of code.**
  - Use \ when must go to next line prematurely.
- **No braces { } to mark blocks of code in Python…**
- **Use *consistent* indentation instead (4 spaces).**
  - The first line with *less* indentation is outside of the block.
  - The first line with *more* indentation starts a nested block.
- **Often a colon : appears at the start of a new block. (E.g. for function and class definitions.)**

Comments

- **Start comments with # and a single space – the rest of line is ignored.**
- **Block comments**
  - Multiple lines starting with # and a space

Basic operators

- **Assignment uses = and comparison uses == and !=**
- **For numbers + - * / % as expected.**
  - Special use of **+** for string concatenation.
  - Special use of **%** for string formatting (as with printf in C)
  - ** for exponentiation
- **Logical operators are words (and, or, not) *not* symbols**
  - **!= for exclusive or**
- **The basic printing command is the print() function (since Python 3 no print operator).**
- **The first assignment to a variable creates it.**
  - Variable types don't need to be declared.
  - Python figures out the variable types on its own.

Basic operators (continued)

- **No increment (++) / decrement (--) operators**
  - Instead: `x = x + 1 or x += 1`
- **Composite assignment operators as expected**
  - **`*=  /=  %=  -=`**

Basic datatypes (Python3)

- **Integers**
  - `z = 5 // 2      # Answer is 2, integer division`
- **Floats**
  - `x = 3.5`
  - `y = 5 / 2       # Answer is 2.5, float division`
  - `z = 5.0 / 2     # Answer is 2.5`
  - `v = 5.0 // 2 # Answer is 2.0, float floor`
- **Booleans**
  - `t = True`
  - `f = False`
- **Strings**
  - `s = "abc"` or `s = 'abc'` (Same thing.)
  - Triple quotes for multi-line strings or strings containing both `'` and `"` inside of them. `"""a'b"c"""`

Naming

- Case-sensitive, may contain letters, numbers, underscores.
- Cannot start with a number
- Cannot use reserved words:

```
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
```

Assignment

- ***Binding a variable*** **in Python means setting a** ***name*** **to hold a** ***reference*** **to some** ***object*****.**
  - *Assignment creates references, not copies*
- **Names in Python do not have an intrinsic type. Objects have types.**
  - Python determines the type of the reference automatically based on the data object assigned to it.
- **You create a name the first time it appears on the left side of an assignment expression:**

```
x = 3
```

- **A reference is deleted via garbage collection after any names bound to it have passed out of scope.**

Assignment (continued)

- **If you try to access a name before it's been properly created (by placing it on the left side of an assignment), you'll get an error.**

```
>>> y
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'y' is not defined
```

- **Multiple Assignment**

```
x, y = 2, 3
```

Strings

- **String assignment sprintf style**

```
hello = '%s %s %d' % ("hello", "world", 12)
print(hello) # prints "hello world 12"
```

- **String objects have a number of useful methods**

```
len(hello) # =14
s = "hello"
print(s.capitalize()) # Capitalize, prints "Hello"
print(s.upper()) # To uppercase; prints "HELLO"
# Replace all instances of one substring with another
print(s.replace('l', '(ll)')) # prints "he(ll)(ll)o"
# Strip leading and trailing whitespace
print(' world '.strip()) # prints "world"
```

Containers: Lists

- **Lists**
  - Equivalent to arrays, but resizeable and can contain elements of different types. Indexing starts from zero.

```
xs = [3, 1, 2]
print(xs, xs[2])
print(xs[-1])  # Negative indices count from the end

[3, 1, 2] 2
2
xs[2] = "foo"
print(xs)      # prints [3, 1, 'foo']
```

Lists (continued)

```
xs.append('bar); # appends a new element at the end
print(xs)         # prints [3, 1, 'foo', 'bar']

x = xs.pop();     # Removes & returns the last element
print(x, xs)      # prints bar [3, 1, 'foo']
```

Assignment (reviewed)

- **Assignment manipulates references**
  - x = y **does not make a copy** of the object y references
  - x = y makes x **reference** the object y references
- **Example**

```
a = [1, 2, 3];    # a now references the list [1,2,3]
b = a             # b references what a references
a.append(4)
print(b)          # prints [1, 2, 3, 4]
```

Assignment: reference semantics

- **There is a lot going on when we type:** `x = 3`
  - First, an integer 3 is created and stored in memory
  - A name `x` is created
  - A *reference* to the memory location storing the `3` is assigned to the name x
- **Integer, float, string (and tuple) data types are immutable.**
  - In order to change the value of `x` we must change what x refers to.

```
x = 3
x = x + 1
print(x)     # prints 4
```

Assignment: reference semantics II

- **If we increment x, then what's really happening is:**
  1. The reference of name ***x*** is looked up.
  2. The value at that reference is retrieved.
  3. The **3+1** calculation occurs, producing a new data element which is assigned to a fresh memory location with a new reference.
  4. The name **x** is changed to point to this new reference.
  5. *The old data **3** is garbage collected if no name still refers to it.*

Assignment: reference semantics III

```
# creates 3, name x refers to 3
x = 3
# creates name y, refers to 3
y = x
# creates 4, new ref of y to 4
y = 4
# no effect to x, still ref to 3
print x
```

```
x = some mutable object
y = x
make a change to y [e.g.
append to a list]
look at x
x will be changed as well
```

Immutable objects

Mutable objects

List slicing

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

List slicing / Loops

```
nums[2:4] = [8, 9]    # Assign a new sublist to a slice
print(nums)           # Prints "[1, 2, 8, 9, 5]"
```

- **Loop over the elements of the list:**

```
simplices = ['point', 'segment', 'triangle', 'tetrahedron']
for simplex in simplices:
    print(simplex)
```

- **Access the index**

```
simplices = ['point', 'segment', 'triangle', 'tetrahedron']
for i, simplex in enumerate(simplices):
    print("%d-simplex %s" % (i, simplex))
```

List comprehensions

```
nums = list(range(1,1000))
squares = [x ** 2 for x in nums]

nums = list(range(1,1000))
even_squares = [x ** 2 for x in nums if x % 2 == 0]
```

Containers: dictionaries

- **A dictionary stores (key, value) pairs, similar to a Map in Java**

```
d = {'0-simplex': 'point', 1: 'segment', 2: 'triangle', 3: \
'tetrahedron'}
print(d['0-simplex']) # prints "point"
print('4-simplex' in d) # checks if '4-simplex' key exists;
                        # prints "False" (True if exists)
```

- **Key-value pairs may be added as necessary**

```
d['4-simplex'] = '5-cell'
print(d['4-simplex'])   # prints '5-cell'
```

- **Attempt to retrieve a non-existent entry: KeyError**

```
print(d['5-simplex'])   # KeyError, instead
print(d.get('5-simplex', 'N/A') # 'N/A' default if not found
```

Dictionaries (continued)

- **Delete a key-value pair**

```
del d['0-simplex']
print(d.get('0-simplex', 'N/A')) # prints 'N/A'
```

- **Iterate over the keys**

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

or

```
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
```

Dictionary comprehensions / Sets

```
nums = list(range(1,1000))
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
```

- **Sets: a set is an unordered collection of distinct elements**

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

Sets (continued)

- **Iterating over a set is the same as iterating over a list**
- **No assumption can be made about the order of the elements**

```
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 are similar to lists and dictionaries**

```
from math import sqrt
print({int(sqrt(x)) for x in range(30)})

# Prints {0, 1, 2, 3, 4, 5}
```

Tuples

- **A tuple is an immutable ordered list of values.**
- **Tuples can be used as keys in dictionaries and as elements in sets while lists cannot.**

```
d = {(x, x+1): x for x in range(10)}
t = (5, 6)
print(d[t])         # prints 5
print(d[(1,2)])     # prints 1
print(t[0])         # prints 5
t[0] = 1            # TypeError (immutable)
```

Functions

- **Functions are defined and assigned a name using the def keyword.**
- **Return types and argument types are not declared.**

```
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\n zero\n positive\n
```

Functions (continued)

- **Arguments are passed by assignment**
- **Passed arguments are assigned to local names**
- **Changing a mutable argument may affect the caller**

```
def changer (x,y):
    x = 2           # changes local value of x only
    y[0] = 'hi'     # changes shared object
```

- **Can be defined with optional arguments**

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

hello('Bob')          # prints 'Hello Bob'
hello('Bob', True)    # prints 'HELLO BOB'
```

Functions (continued)

- All functions in Python have a return value
  - even if no return statement inside the code
- Functions without explicit return, return the special value *None.*
- Two different functions cannot have the same name
- Functions can be used as any other data type
  - Arguments to functions
  - Return values of functions
  - Assigned to variables
  - Parts of tuples, lists, etc.

Flow control

- if *condition* / elif *condition* / else
- assert(*condition*)
- for *variable* in *container*
- while *condition*
- break / continue

Modules

- **Modules are functions and variables defined in separate files**
- **Items are imported using from or import**

```
from module import function
function()

import module
module.function()
```

- **Modules are namespaces and can be used to organize names, e.g. np.array**

Classes

- **Easy syntax for the definition of classes**

```
class Greeter:

    #Constructor
    def __init__(self,name):
        self.name = name

    #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 class
g.greet()            # prints 'Hello Fred'
g.greet(True)        # prints 'HELLO FRED'
```

Numpy: Arrays

- Numpy is the core Python library for scientific computing
- Provides a high-performance multi-dimensional array object and tools for working with the arrays.
- Numpy should be imported in the code before use.

```
import numpy as np
```

- Numpy arrays are initialized from nested Python lists.
- Elements are accessed with square brackets.
- Example with an 1-dimensional array:

```
a = np.array([1, 2, 3])              # create a rank 1 array
print(a.shape, a[0], a[1], a[2])     # prints (3,) 1 2 3
a[0] = 5
print(a)                             # prints [5 2 3]
```

Arrays (continued)

- **Examples with 2-dimensional arrays:**

```
b = np.array([[1, 2, 3],[4, 5, 6]])   # create a rank 2 array
print(b)                              # prints [[1 2 3]
                                      #         [4 5 6]]
print(b.shape)                        # prints (2,3)
print(b[0, 0], b[0, 1], b[1, 0])      # prints 1 2 4

a = np.zeros((2,2))                   # creates 2*2 array with zeros
c = np.ones((1,2))                    # creates 1*2 array with ones
print(c)                              # prints [[1. 1.]]
d = np.ones((2,))                     # created 2*1 array with ones
print(d)                              # prints [1. 1.]
a = np.full((2,2),7)                  # creates a constant 2*2 array
c = np.eye(2)                         # creates a 2*2 identity matrix
print(c)                              # prints [[1. 0.]
                                      #          [0. 1.]]

x = np.arange(0, 10, 0.1)             # construct an array from 0 to
                                      # to 10 with step 0.1
```

Arrays (continued)

- **Examples with 2-dimensional arrays:**

```
c = np.random.random((2,2))   # create a 2*2 array with random values
```

- **Array indexing and slicing:**

```
a = np.array([[1, 2, 3, 4],[5, 6, 7, 8], [9, 10, 11, 12]])
b = a[:2, 1:3]
print(b)                      # prints [[2 3]
                              #          [6 7]]
```

- **A slice is a view of the same data:**

```
b[1, 0] = 40
print(a[1,1])                 # prints 40
```

- **Enforce a particular datatype of the data**

```
x = np.array([1,2], dtype=np.int64) # Data types: int64, float64
                                    # float32, int32, uint64,
                                    # complex64, complex128, etc.
                                    # otherwise numpy decides
```

Array math

- **Array addition / difference / product / division [element-wise]**

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
c = x + y
c = np.add(x,y)              # equivalent
print(c)                     # prints [[6. 8.]
                             #         [10. 12.]]
c = x - y
c = np.subtract(x, y)        # equivalent
c = x * y
c = np.multiply(x, y)        # equivalent
c = x / y
c = np.divide(x, y)          # equivalent
c = np.sqrt(x)               # element-wise square root of x
```

Dot product / Matrix multiplication

- **Both performed by the dot function of numpy**
- **Dot product of vectors**

```
v = np.array([9, 10])
w = np.array([2, 1])
print(np.dot(v, w))                 # prints 28
print(v.dot(w))                     # equivalent
```

- **Matrix / vector product**

```
x = np.array([[1, 2], [3, 4]])
w = np.array([2, 1])
print(np.dot(x, w))                 # prints [4 10]
print(x.dot(w))                     # equivalent
p = x.dot(w)
print(p.shape)                      # prints (2,)
```

Matrix multiplication

- **Matrix / matrix product**

```
x = np.array([[1, 2], [3, 4]])
y = np.array([[2, 1], [0, 1]])
print(np.dot(x, y))                 # prints [[2 3]
                                    #          [6 7]]
print(x.dot(w))                     # equivalent
p = x.dot(w)
print(p.shape)                      # prints (2,2)
```

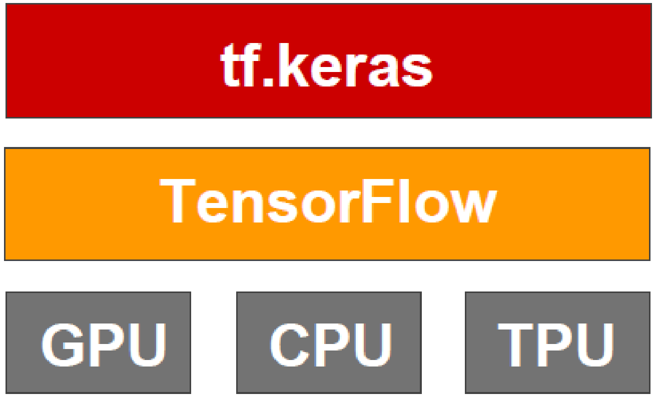
- **Matrix transpose**

```
x = np.array([[1, 2], [3, 4]])
y = x.T
print(y)                            # prints [[1 3]
                                    #          [2 4]]
```

- **Broadcasting:** consult *Python Documentation*

Keras API

- A high-level programming interface for the Neural Network library TensorFlow
- Enables to easily and quickly build and train neural nets with multiple layers



TensorFlow & Keras Installation

- $ pip3.7 install --user tensorflow
- $ pip3.7 install --user pandas [βιβλιοθήκη για την εύκολη επεξεργασία αρχείων CSV]
- $ pip3.7 install --user keras

1-1. Build the NN with Sequential API

- **The Sequential Model / API is the simplest approach to build a Neural Net with multiple layers (good for > 70% of use cases).**

Build the NN

```
import keras
import numpy as np
import pandas as pd
from keras import layers, optimizers, losses, metrics
# Initializes the model
model = keras.Sequential()
# Adds a densely-connected layer with 64 nodes to the model:
# input shape defines the number of input features (dimensions)
# activation defines the activation function of each layer
model.add(layers.Dense(64, activation='relu', input_shape=(10,)))
# Add another:
model.add(layers.Dense(64, activation='relu'))
# Add a softmax layer with 10 output nodes:
model.add(layers.Dense(10, activation='softmax'))
```

1-2. Build the NN details

- **Activation function of the last layer**
  - Define according to the problem type:
    - softmax (classification into multiple classes)
    - linear (regression)
    - sigmoid (binary classification)
- **Number of output nodes**
  - Define according to the specific problem:
    - Binary Classification: `model.add(layers.Dense(2, activation='sigmoid'))`
    - Categorical Classification: `model.add(layers.Dense(10, activation='softmax'))` `[10 classes]`
    - Multi-dimensional Regression: `model.add(layers.Dense(10, activation='linear')) [vector with 10 dimensions]`

2. Compile the NN

- **Code continues from slide 52**

```
model.compile(optimizer=optimizers.RMSprop(0.01),
              loss=losses.CategoricalCrossentropy(),
              metrics=[metrics.CategoricalAccuracy()])
```

- **Select optimizer according to the problem type. E.g:**
  - RMSprop (Root Mean Square propagation) for categorical classification
  - Adam (Adaptive Moment Estimation) for regression
  - The first parameter (e.g. `0.01`) is the learning rate
- **Select loss function according to the problem type. E.g.:**
  - CategoricalCross Entropy for categorical classification
  - mse (Mean Squared Error) for regression
- **The metrics parameter defines which evaluation functions will be calculated to judge the performance of the model.**

3-1. Train the NN

- **Code continues from slide 54**

```
# The number of columns of the input must be equal with the number of
# rows of the input shape of the first layer. The number of columns of
# the "labels" must be equal with the number of the output nodes.
# The number of rows of data and labels is the size of the training set.
data = np.random.random((1000, 10))
labels = np.random.random((1000, 10))

model.fit(data, labels, epochs=10, batch_size=100)
```

- **The batch size is a hyperparameter of gradient descent (or other optimizer) that controls the number of training samples to work through before the model's internal parameters are updated.**
- **The number of epochs is a hyperparameter of gradient descent (or other optimizer) that controls the number of complete passes through the training dataset**

3-2. Train the NN with input from CSV

```
# in place of the corresponding lines of code
data = pd.read_csv('data.csv')
labels = pd.read_csv('labels.csv')
```

4. Evaluate & Predict in inference of provided data

- **Code continues from slide 55**

```
# Evaluate prediction according to defined metrics (slide 53)
model.evaluate(data, labels, batch_size=32)

# Get a random data set for prediction
test_data = np.random.random((1000, 10))
result = model.predict(test_data, batch_size=32)
print(result.shape)
print(result)
```

Load and process a pre-trained model

- **A pre-trained model is saved in .h5 binary format (Hierarchical Data Format) and can be loaded as follows:**

```
from keras.models import load_model

model = load_model('pathtomodel.h5')
# summarize the structure of the model
model.summary()

# Get the weights of the first layer of the model
model.layers[0].get_weights()
```

Matplotlib

- **Matplotlib is a plotting library. Import as follows:**

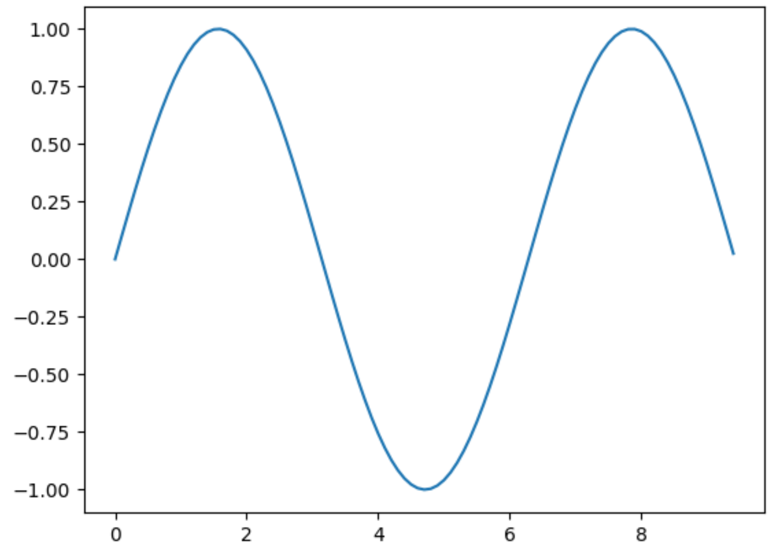
```
import matplotlib.pyplot as plt
```

- **Script must be run with ipython**

```
$ipython plot.py
```

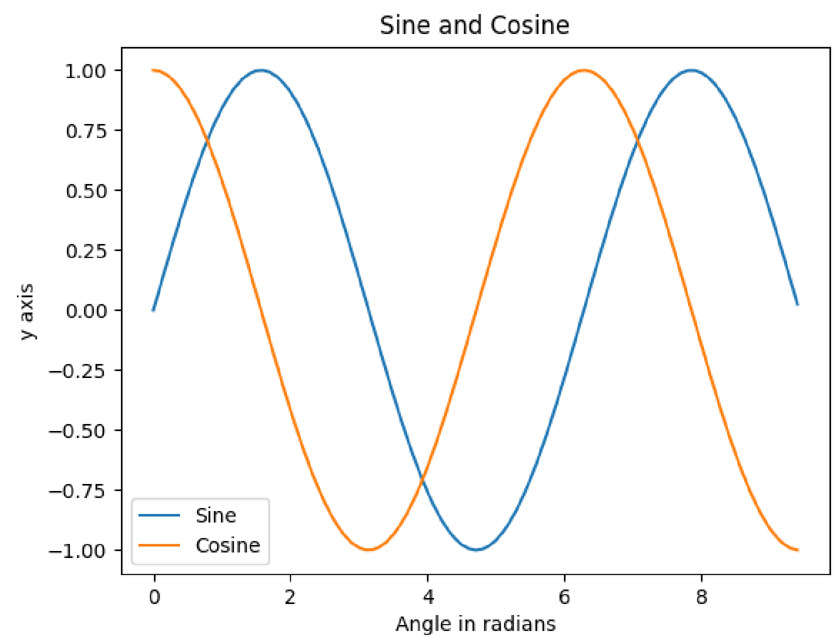
- **Example**

```
x = np.arange(0, 3*np.pi, 0.1)
y = np.sin(x)

#Plot the points
plt.plot(x,y)
plt.show()    #in Ubuntu Linux
```



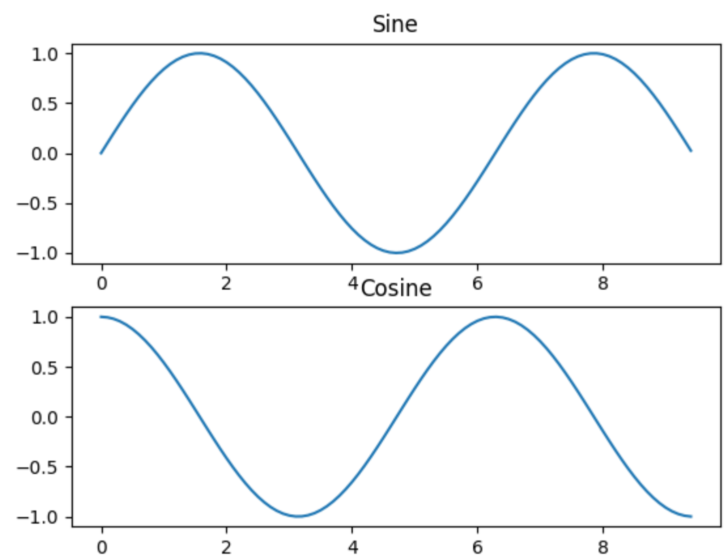
Multiple plots

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 3*np.pi, 0.1)
y = np.sin(x)
y_cos = np.cos(x)

plt.plot(x, y)
plt.plot(x,y_cos)
plt.xlabel('Angle in radians')
plt.ylabel('y axis')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



Subplots

```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)
# Set up a subplot grid that has height 2 and width 1 subplots,
# and set the first such subplot as active. Index starts at 1
in the upper left corner and increases to the right.
plt.subplot(2, 1, 1)
plt.plot(x, y_sin)
plt.title('Sine')
# Set the second subplot as active
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')
plt.show()
```



Scikit-learn

- **Scikit-learn is the standard library for machine learning in Python.**
- **Nearest neighbor search**
  - **Supported algorithms: K-D Tree, Ball Tree, Brute Force**

```
from sklearn.neighbors import NearestNeighbors as nn
import numpy as np
# Data Set
X = np.array([[-1,-1],[-2,-1],[-3,-2],[1, 1],[2, 1],[3, 2]])
# Query Set
Y= np.array([[0,1], [2, 3]])
nbrs = nn(n_neighbors=2, algorithm='ball_tree').fit(X)
distances, indices = nbrs.kneighbors(Y)
print(indices)
print(distances)
```

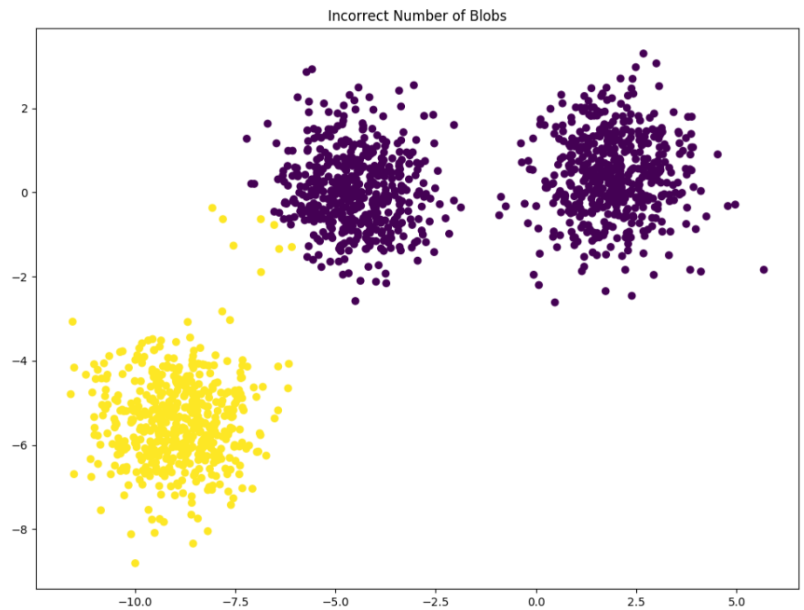
Nearest Neighbors

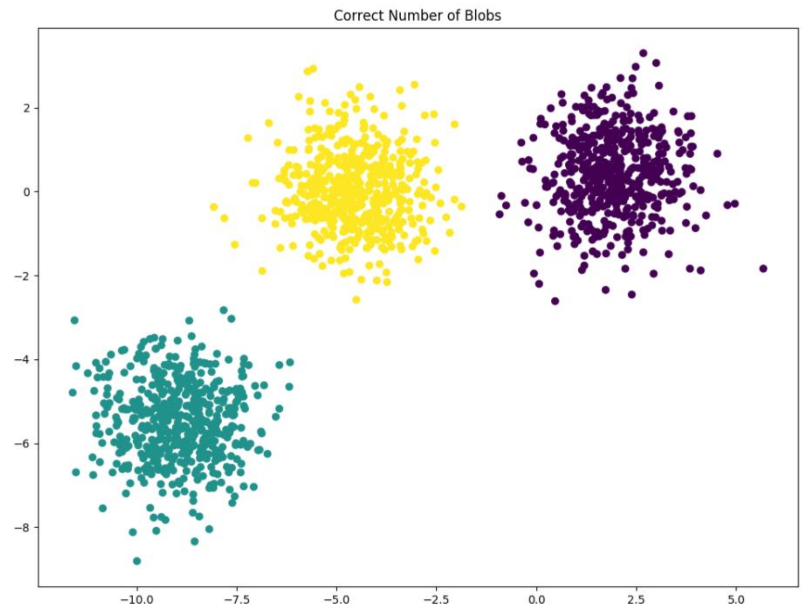
- **indices**: array of the indices of the k nearest neighbors of each point in the query set. In the example: `[[3 4] [5 4]]`.
- **distances**: array of the distances of the k nearest neighbors of each point in the query set.
  In the example: `[[1. 2.] [1.41421356 2.]]`
- If the query set is not defined in **kneighbors** function, the neighbors of each indexed point in the dataset are returned. In this case, the query point is not considered its own neighbor.
- **Further documentation:**
  - https://scikit-learn.org/stable/modules/neighbors.html

k-Means Clustering

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

plt.figure(figsize=(12, 12))

n_samples = 1500
random_state = 170 # seed for gaussian blobs, default centers = 3, default features = 2
X, y = make_blobs(n_samples=n_samples, random_state=random_state)

# y_pred array of indices of each sample to each cluster
# Incorrect number of clusters, should be n_clusters = 3
# random_state is the seed for Kmeans initialization
y_pred = KMeans(n_clusters=2, random_state=random_state).fit_predict(X)

# arrays for each feature of the samples
# c for color from cluster index
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.title("Incorrect Number of Blobs")
plt.show()
```

k-Means Clustering (continued)





Further reading

- https://docs.python.org/3/
- https://docs.scipy.org/doc/
- https://www.tensorflow.org/guide/keras
- https://matplotlib.org/contents.html
- https://scikit-learn.org/stable/user_guide.html