

Python Tutorial

Fall 2019

Xishuang Dong, PhD.

**Assistant Professor
Department of Electrical and Computer Engineering
Roy G. Perry College of Engineering
Prairie View A&M University**

11/08/2019

Acknowledgement

- Some of the slides or images are from various sources. The copyright of those materials belongs to their original owners.

Outline

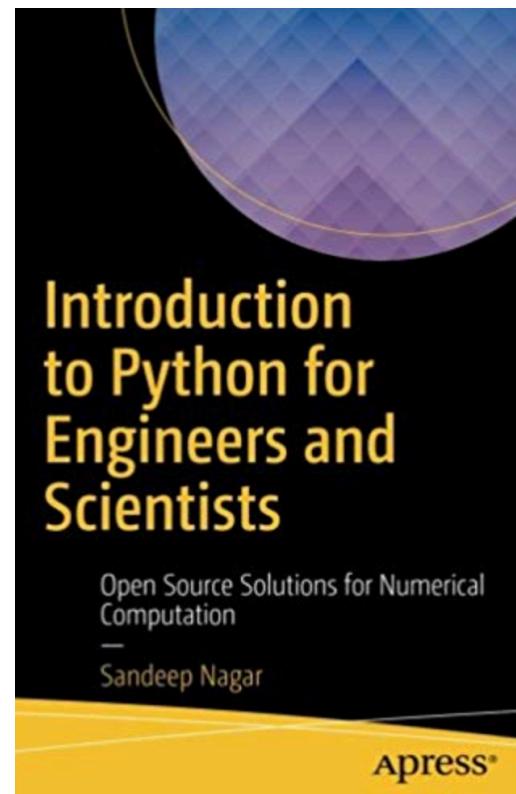
- **Fundamentals**
- **Introduction to Data Structures**
- **Programming Structure and Functions**
- **Python Standard Library**
- **Advanced Topics**
- **Projects**

Outline

- **Fundamentals**
- Introduction to Data Structures
- Programming Structure and Functions
- Python Standard Library
- Advanced Topics
- Projects

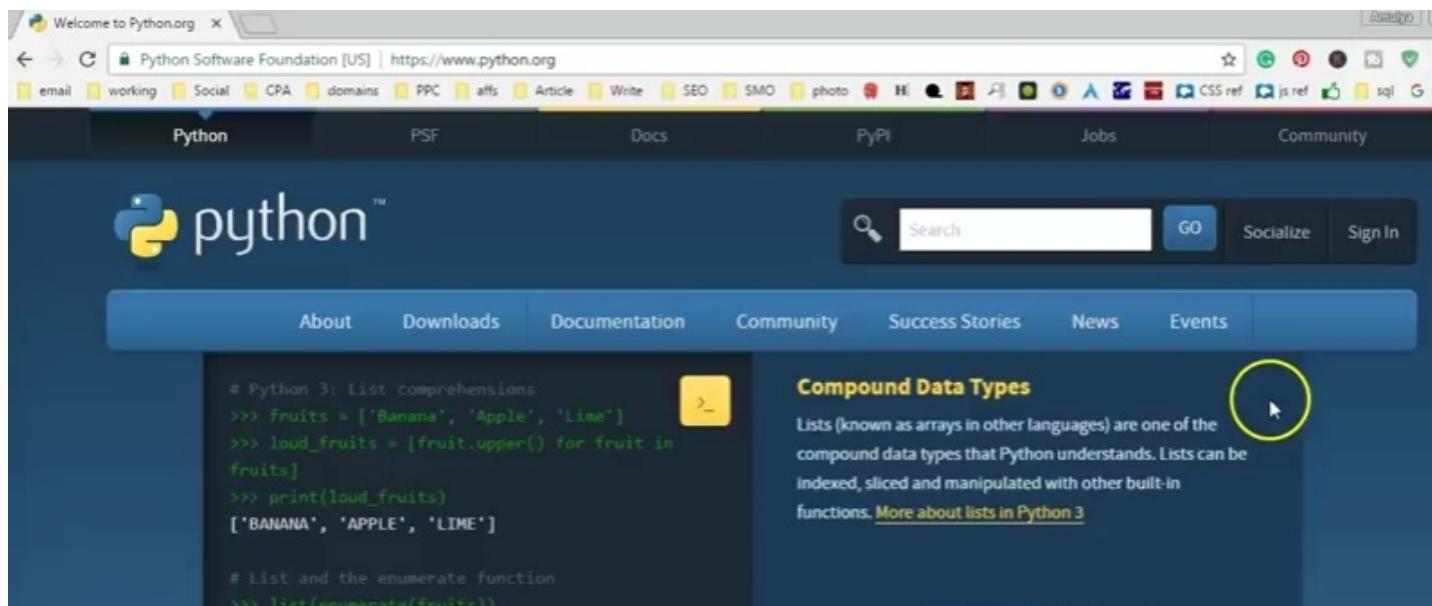
Why Python?

- Python is a powerful, multi-paradigm, interpreted language popular.



Installation

- Windows

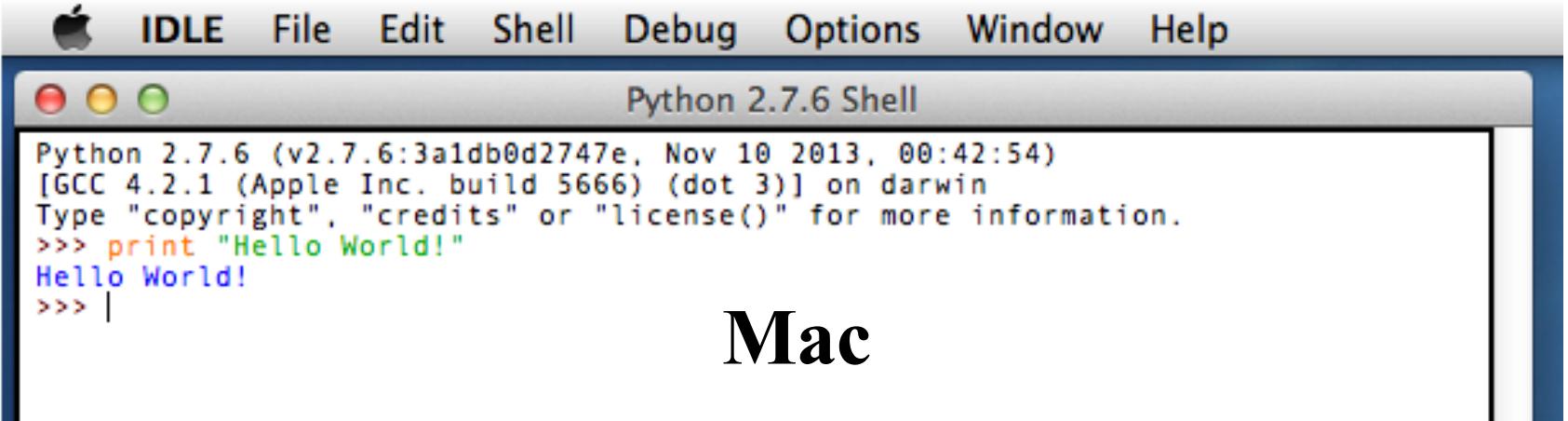


https://www.youtube.com/watch?v=oHOiqFs_x8Y

- How to install Python on Windows
(<https://www.howtogeek.com/197947/how-to-install-python-on-windows/>)

Installation

- Mac & Linux (Ubuntu)
 - Prepackaged Python 2

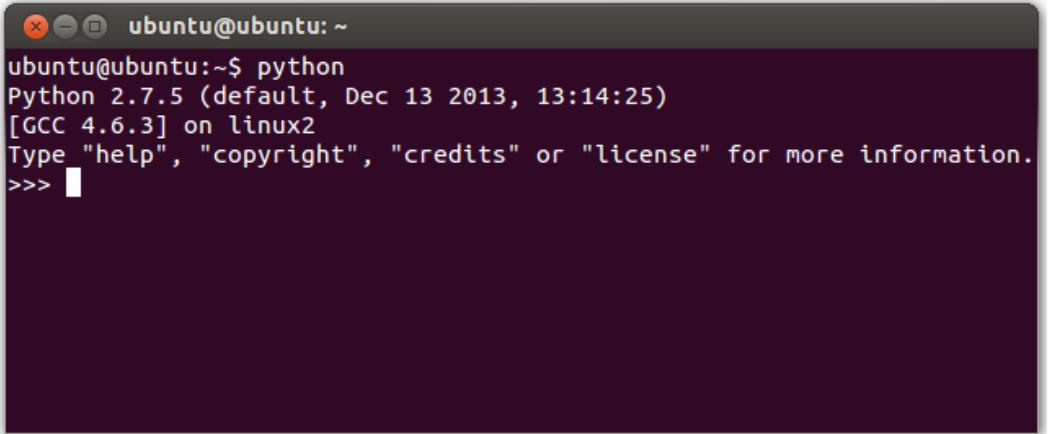


The screenshot shows the Python 2.7.6 Shell window running on a Mac OS X desktop. The window title is "Python 2.7.6 Shell". The menu bar includes "IDLE", "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main window displays the following Python session:

```
Python 2.7.6 (v2.7.6:3a1db0d2747e, Nov 10 2013, 00:42:54)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print "Hello World!"
Hello World!
>>> |
```

A large "Mac" watermark is overlaid on the bottom left of the window.

Linux (Ubuntu)



The screenshot shows a terminal window on an Ubuntu system. The title bar says "ubuntu@ubuntu: ~". The command entered is "python". The terminal displays the following Python session:

```
ubuntu@ubuntu:~$ python
Python 2.7.5 (default, Dec 13 2013, 13:14:25)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

Python 2 vs Python 3

- For beginners there is no real difference between Python 2 & 3. The basics are the same.



Python 2 vs Python 3

- Examples are written for Python2.
- Python2 will be maintained until 2020.



Python 2 vs Python 3

- **Python 2**

```
>> print ("hello world")
```

```
>> print "hello world"
```

- **Python 3**

```
>> print ("hello")
```

```
>> print "hello"
```

```
File "<input>", line 1
    print "hello"
               ^
SyntaxError: invalid syntax
```

Python 3 HELLO WORLD

- *print* is no longer a statement, but a FUNCTION

```
print("hello world")
```

From Script

- Make a file `hello.py` with
print “hello world”
- Run with:
python hello.py

Python syntax

Python

```
x = 5  
y = x + 7  
z = 3.14
```

```
name = "Rishi"
```

```
1 == 1 # => True  
5 > 10 # => False
```

```
True and False # => False  
not False # => True
```

Variables are not
statically typed!

Java/C++

```
int x = 5;  
int y = x + 7;  
double z = 3.14;
```

```
String name = "Rishi"; // Java  
string name("Rishi"); // C++
```

```
1 == 1 # => true  
5 > 10 # => false
```

```
true && false # => false  
!(false) # => true
```

Variables

```
a = 4          # Integer
b = 5.6        # Float
c = "he11o"    # String
a = "4"        # rebound to String
```

Naming

- **Lowercase**
- **Underscore_between_words**
- **Don't start with numbers**

Math

- `+, -, *, /, ** (power), % (modulo)`
- Careful with integer division

```
»»» 3/4
```

```
0
```

```
»»» 3/4.
```

```
0.75
```

Data Type

- Long

```
>>> 10**100
```

```
10000000000000000000000000000000000000000000  
0000000000000000000000000000000000000000000  
0000000000000000000000000000000000000000000  
00000L
```

```
>>> import sys
```

```
>>> sys.maxint
```

```
9223372036854775807
```

```
>>> sys.maxint + 1
```

```
9223372036854775808L
```

Strings

```
name = 'matt'  
with_quote = "I ain't gonna"  
longer = """This string has  
multiple lines  
in it"""
```

String escaping

- Escape with \

```
>>> print 'He said, "I\'m sorry"'  
He said, "I'm sorry"  
>>> print '''He said, "I'm sorry'''  
He said, "I'm sorry"  
>>> print """He said, "I'm sorry\""""  
He said, "I'm sorry"
```

String escaping

Escape Sequence	Output
\\	Backslash
\'	Single quote
\"	Double quote
\b	ASCII Backspace
\n	Newline
\t	Tab

String methods

s.endswith(sub)

Returns True if endswith sub

s.find(sub)

Returns index of sub or -1

Comments

- **Comments follow a #**
- **No multi-line comments**

```
#hello world  
print("hello")
```

Outline

- Fundamentals
- **Introduction to Data Structures**
- Programming Structure and Functions
- Python Standard Library
- Advanced Topics
- Projects

Data Structures

- Lists
- Dictionaries
- Iteration
- List Comprehensions

Lists

- Hold sequences.
- How would we find out documentation for a method?

help function:

```
>>> help([] .append)
Help on built-in function append:
```

```
append(...)
L.append(object) -- append object to end
```

List methods

l.append(x)

Insert x at end of list

l.extend(l2)

Add l2 items to list

l.sort()

In place sort

List methods

`l.reverse()`

Reverse list in place

`l.remove(item)`

Remove first item found

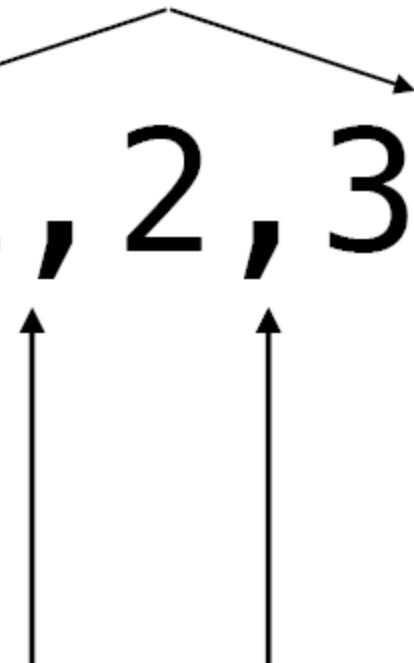
`l.pop()`

Remove/return item at end of list

Lists

Square brackets delimit lists

```
easy_as = [1, 2, 3]
```



Commas separate elements

Lists

```
# Create a new list
empty = []
letters = ['a', 'b', 'c', 'd']
numbers = [2, 3, 5]

# Lists can contain elements of different types
mixed = [4, 5, "seconds"]

# Append elements to the end of a list
numbers.append(7)    # numbers == [2, 3, 5, 7]
numbers.append(11)   # numbers == [2, 3, 5, 7, 11]
```

Lists

```
# Access elements at a particular index
numbers[0] # => 2
numbers[-1] # => 11

# You can also slice lists - the same rules apply
letters[:3] # => ['a', 'b', 'c']
numbers[1:-1] # => [3, 5, 7]

# Lists really can contain anything - even other lists!
x = [letters, numbers]
x # => [['a', 'b', 'c', 'd'], [2, 3, 5, 7, 11]]
x[0] # => ['a', 'b', 'c', 'd']
x[0][1] # => 'b'
x[1][2:] # => [5, 7, 11]
```

Tuples

Parentheses delimit tuples

my_tup = (1, 2, 3)

Like lists, but immutable
my_tup[0] = 5 => Error!

Commas separate elements

Dictionaries

```
dict = {'a': 2,  
        'b': 3}
```

Key

Value

Commas separate entries

Dictionaries

- Also called hashmap or associative array elsewhere

```
>>> age = {}
>>> age['george'] = 10
>>> age['fred'] = 12
>>> age['henry'] = 10
>>> print age['george']
10
```

Dictionaries

- Find out if 'matt' in age

```
>>> 'matt' in age  
False
```

- .get method

```
>>> print age['charles']  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
KeyError: 'charles'  
>>> print age.get('charles', 'Not found')  
Not found
```

Deleting Keys

- Removing 'charles' from age

```
>>> del age['charles']
```

Dictionaries

```
d = {"one": 1, "two": 2, "three": 3}
```

```
len(d.keys()) # => 3
```

```
print d['one'] # => 1
```

```
print d['five'] # => ERROR!
```

```
d['five'] = 5 # => OK, creates new key
```

```
d.keys() # iterator over k
```

```
d.values() # iterator over v
```

```
d.items() # iterator over (k, v) pairs
```

Iteration

- Most data structures can be iterated over in the same way:

```
mylist = ['a', 'b', 'c']  
for item in mylist:  
    print item
```

```
dict = {'a': 10, 'b': 15}  
for key in dict:  
    print key, dict[key]
```

When iterating over a dictionary like this, we iterate over the keys.

```
mytuple = ('a', 'b', 'c')  
for item in mytuple:  
    print item
```

Note we don't need the index of the element to access it.

Iteration

- We can also iterate over indices:

```
for i in range(4):                      # 0 1 2 3
    print i,
    
for i in range(1, 10, 2):                 # 1 3 5 7 9
    print i,
    
mylist = ['a', 'b', 'c']                  # 0 'a'
for i in range(len(mylist)):              # 1 'b'
    print i, mylist[i]                   # 2 'c'

mylist = ['a', 'b', 'c']                  # 0 'a'
for idx, item in enumerate(mylist):       # 1 'b'
    print idx, item                      # 2 'c'
```

List Comprehensions

Input: `nums = [1, 2, 3, 4, 5]`

Goal: `sq_nums = [1, 4, 9, 16, 25]`

Here's how we could already do this:

```
sq_nums = []
for n in nums:
    sq_nums.append(n**2)
```

Or... we could use a comprehension:

```
sq_nums = [n ** 2 for n in nums]
```

square brackets show
we're making a list

apply some operation
to the loop variable

loop over the specified
iterable

Outline

- Fundamentals
- Introduction to Data Structures
- Programming Structure and Functions
- Python Standard Library
- Advanced Topics
- Projects

Programming Structure

- Conditionals

```
if grade > 90:  
    print "A"  
elif grade > 80:  
    print "B"  
elif grade > 70:  
    print "C"  
else:  
    print "D"
```

Comparison Operators

- Supports ($>$, \geq , $<$, \leq , $==$, $!=$)

```
>>> 5 > 9
```

```
False
```

```
>>> 'matt' != 'fred'
```

```
True
```

Logical operators

- and, or, not (for logical)

```
>>> x = 5
```

```
>>> x < -4 or x > 4
```

```
True
```

Iteration (Loop)

```
for number in [1,2,3,4,5,6]:  
    print number
```

```
for number in range(1, 7):  
    print number
```

Range note

- Python tends to follow half-open interval ([start,end)) with range and slices.
 - $\text{end} - \text{start} = \text{length}$
 - easy to concat ranges w/o overlap

Iteration (2)

- Java/C-esque style of object in array access (BAD):

```
animals = ["cat", "dog", "bird"]
for index in range(len(animals)):
    print index, animals[index]
```

- If you need indices, use enumerate If you need indices, use enumerate

```
animals = ["cat", "dog", "bird"]
for index, value in enumerate(animals):
    print index, value
```

break

- Can **break** out of nearest loop

```
for item in sequence:  
    # process until first negative  
    if item < 0:  
        break  
    # process item
```

continue

- Can **continue** to skip over items

```
for item in sequence:  
    if item < 0:  
        continue  
    # process all positive items
```

Iteration (3)

- Can loop over lists, strings, iterators, dictionaries ... sequence like things:

```
my_dict = { "name": "matt", "cash": 5.45}
for key in my_dict.keys():
    # process key

for value in my_dict.values():
    # process value

for key, value in my_dict.items():
    # process items
```

Null Operation

- *pass* is a null operation

```
for i in range(10):  
    # do nothing 10 times  
    pass
```

Functions

```
def fn_name(param1, param2):  
    value = do_something()  
    return value
```

```
def isEven(num):  
    return (num % 2 == 0)  
  
myNum = 100  
if isEven(myNum):  
    print str(myNum) + " is even"
```

- `def` starts a function definition
- `return` is optional
 - if either return or its value are omitted, implicitly returns `None`
- Parameters have no explicit types

Functions

- **def**

function name

(parameters)

: + indent

optional documentation

body

return

Whitespace

- Instead of { use a : and indent.
consistently (4 spaces)

```
def add_2(num):  
    """ return 2  
    more than num  
    """  
  
    return num + 2
```

Default parameters

```
def add_n(num, n=3):  
    """default to  
    adding 3"""  
    return num + n
```

```
five = add_n(2)  
ten = add_n(15, -5)
```

__doc__

- Functions have docstrings. Accessible via `.__doc__` or help via `.__doc__` or help

```
>>> def echo(txt):
...     "echo back txt"
...     return txt
>>> help(echo)
Help on function echo in module __main__:
<BLANKLINE>
echo(txt)
    echo back txt
<BLANKLINE>
```

Putting Functions and List Comprehensions Together

- Goal: given a list of numbers, generate a list that contains **True** for every **even number** and **False** for every **odd number**

```
def isEven(num):  
    return (num % 2 == 0)
```

```
numbers  = [5, 18, 7, 9, 2, 4, 0]  
isEvens  = [isEven(num) for num in numbers]  
## isEvens = [False, True, False, False, True, True, True]
```

Outline

- Fundamentals
- Introduction to Data Structures
- Programming Structure and Functions
- **Python Standard Library**
- Advanced Topics
- Projects

Importing Modules

```
from math import exp  
from random import random
```

Imports only the selected function(s) from the module; in this case, exp from math and random from random

Can now do:

`exp(0.5)` to compute $e^{0.5}$

`random()` to generate uniform random number over $[0,1]$

But if we had imported the whole modules, like this...

```
import math  
import random
```

Then we would call the functions like this:

```
math.exp(0.5)  
random.random()
```

Numeric Computing using

NumPy

- Python's built-in datatypes are very flexible
- They aren't optimized for fast numerical calculations, especially on large multidimensional matrices
- NumPy is a widely-used 3rd part package which adds such support to Python
- Sister library for scientific computing: SciPy

NumPy

```
>>> import numpy as np  
>>> mat = np.ones((3,3))  
>>> print mat  
[[ 1.  1.  1.]  
 [ 1.  1.  1.]  
 [ 1.  1.  1.]]  
>>> mat[1,1] = 5  
>>> print mat  
[[ 1.  1.  1.]  
 [ 1.  5.  1.]  
 [ 1.  1.  1.]]  
>>> vec = np.array([1, 2, 3])  
>>> np.dot(mat, vec)  
array([ 6., 14.,  6.])
```

I can rename my module when I import it for convenience

It looks a lot like a list of lists!

Create arrays using np.array

Support for various linear algebra operations like dot products

Outline

- **Fundamentals**
- **Introduction to Data Structures**
- **Programming Structure and Functions**
- **Python Standard Library**
- **Advanced Topics**
- **Projects**

Advanced Topics

- File IO
- Class

File Input

- Open a file to read from it:

```
fin = open("foo.txt")
for line in fin:
    # manipulate line
```

```
fin.close()
```

File Output

- Open a file using 'w' to write to a file:

```
fout = open("bar.txt", "w")
fout.write("hello world")
fout.close()
```

File IO

Always remember to close
your files!

Class

```
class Animal(object):
    def __init__(self, name):
        self.name = name

    def talk(self):
        print "Generic Animal Sound"

animal = Animal("thing")
animal.talk()
```

Classes (2)

- notes:
 - object (base class) (fixed in 3.X)
 - Default init (constructor)
 - All methods take self as first parameter

Subclass

```
class Cat(Animal):
    def talk(self):
        print '%s says, "Meow!"' % (self.name)

cat = Cat("Groucho")
cat.talk() # invoke method
```

Outline

- **Fundamentals**
- **Introduction to Data Structures**
- **Programming Structure and Functions**
- **Python Standard Library**
- **Advanced Topics**
- **Projects**

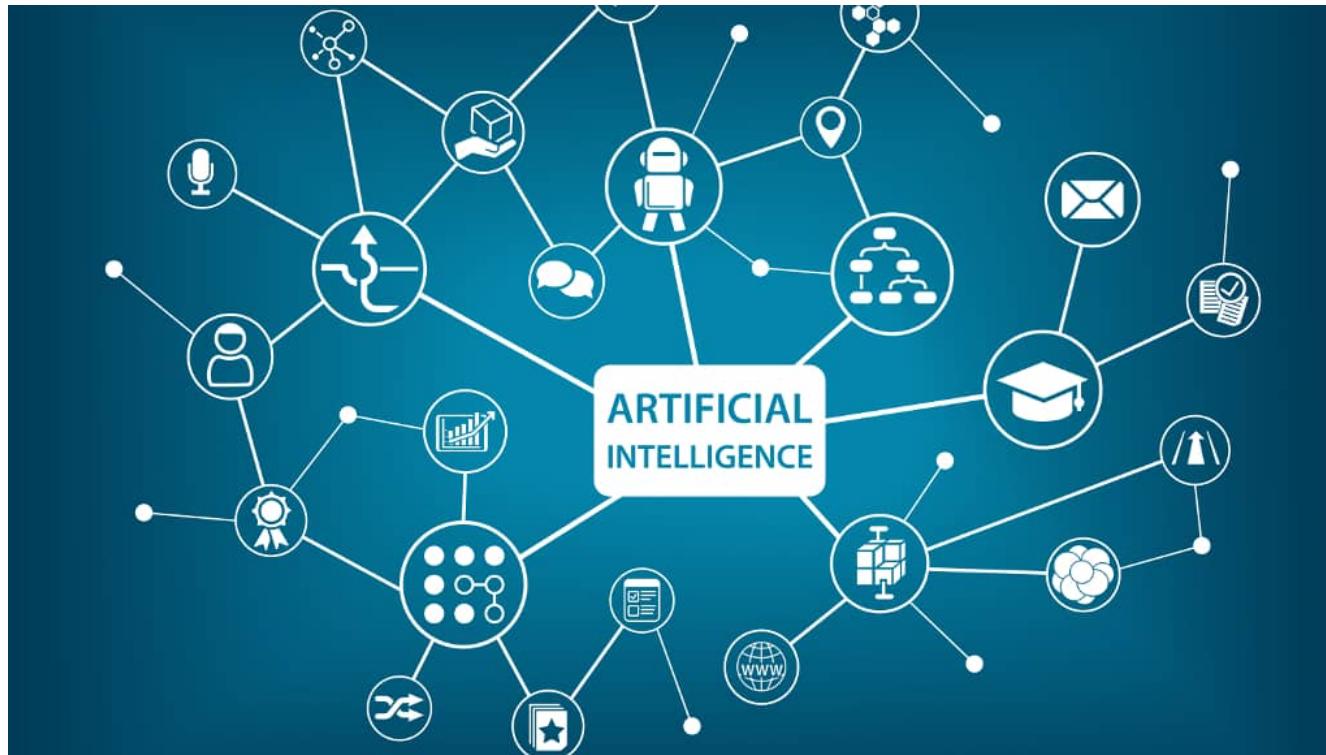
Project

- Automatic Diagnosis on Diabetes



<http://theindianpractitioner.com/2019/07/01/nasscom-and-nathealth-collaborate-to-boost-indias-digital-health/>

Artificial Intelligence



<https://www.gettingsmart.com/2018/12/the-future-is-here-artificial-intelligence-what-it-means-for-our-kids-2/>

Machine Learning

The infographic is divided into three main sections by color: teal, light blue, and light teal. Each section contains a title, a definition, and an icon. The teal section on the left contains the title 'ARTIFICIAL INTELLIGENCE' with a gear and profile icon, followed by a definition of AI as a field of science concerned with getting computers to do tasks requiring human intelligence. The light blue section in the middle contains the title 'MACHINE LEARNING' with a gears and book icon, followed by a definition of ML as a set of algorithms allowing computers to learn from data without explicit programming. The light teal section on the right contains the title 'DEEP LEARNING' with a brain and circuit board icon, followed by a definition of DL as a more recently developed set of learning techniques. A small photo credit at the bottom right states 'Photo Credit: From Noun Project by ImageCatalog, Beavis and DigiPhus'.

ARTIFICIAL INTELLIGENCE

A field of science that is primarily concerned with getting computers to do tasks that would normally require human intelligence.

MACHINE LEARNING

A set of algorithms that allow computers to learn from data without being explicitly programmed

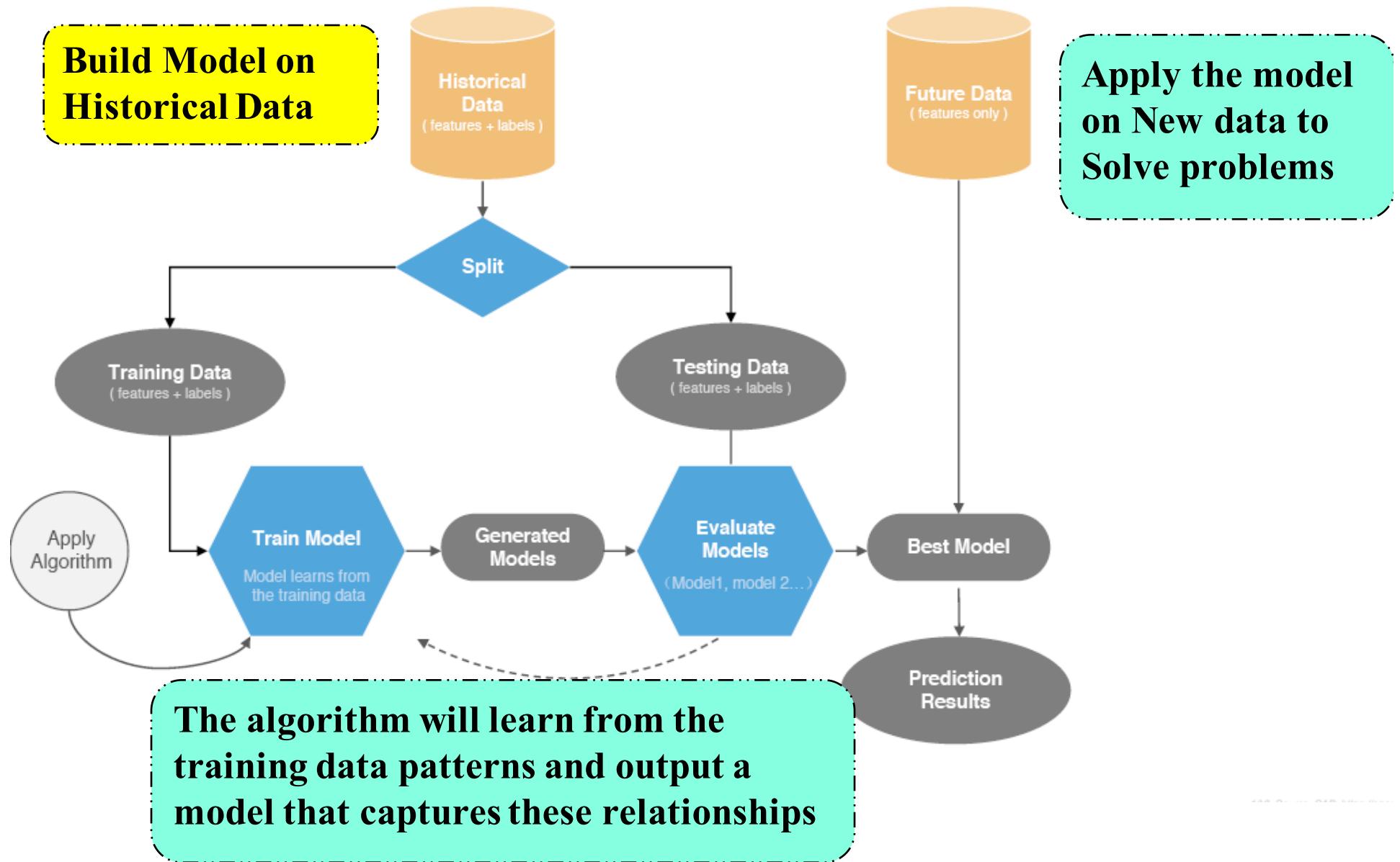
DEEP LEARNING

A more recently developed set of learning techniques

Photo Credit: From Noun Project by ImageCatalog, Beavis and DigiPhus

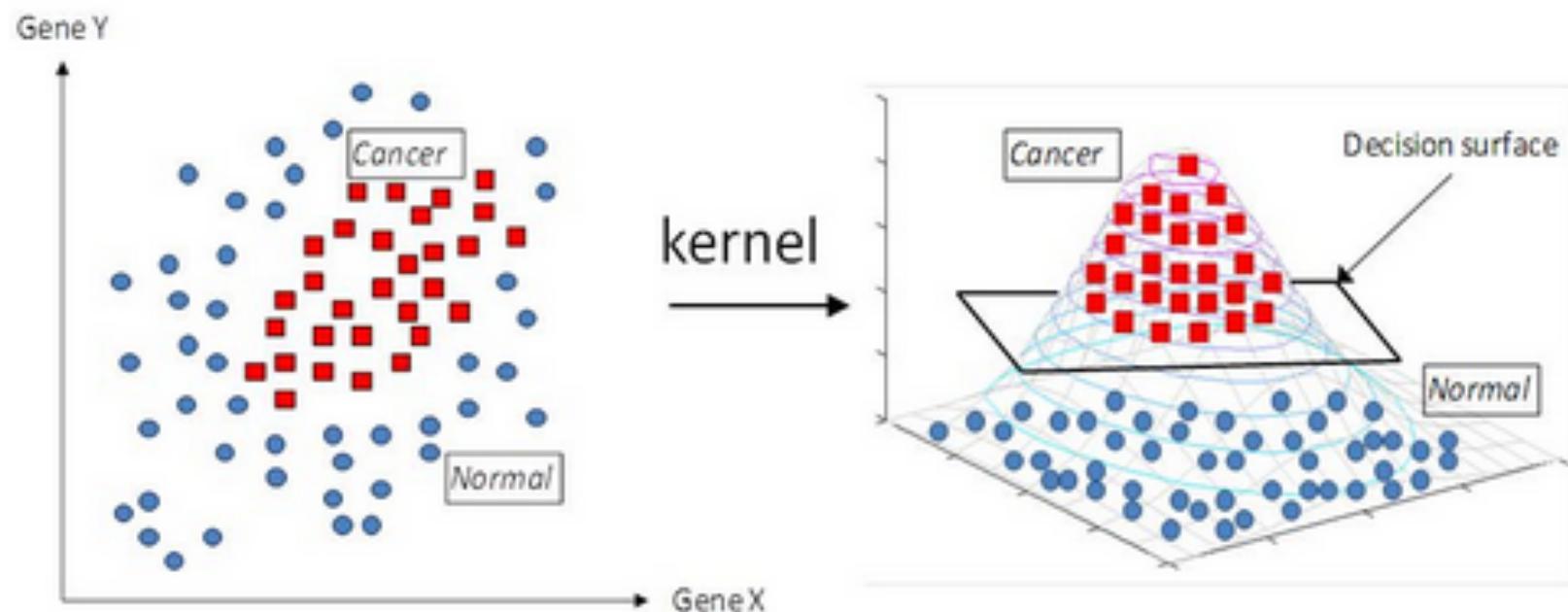
<https://connect.bim360.autodesk.com/what-is-machine-learning-in-construction>

Flow



Support Vector Machine (SVM)

- Supervised learning
- Mapping from low-dimensional space to high-dimensional space with **kernel function**



Feature Selection

- Feature selection is the process of selecting **a subset of relevant features** (variables, predictors) for model construction.



Data

- Pima Indians Diabetes Dataset
(<https://www.andreagrandi.it/2018/04/14/machine-learning-pima-indians-diabetes/>)
 - **768 observations with 8 input variables and 1 output variable**
 - This dataset is known to have missing values.

Python Tutorial

Q & A