DEEP LEARNING & NEURAL NETWORKS

3 MONTHS COURSE

AIE
ACADEMY OF INNOVATIVE EDUCATION

SPEAKER

Thakshila Dasun

BSc. Engineering Hons, CIMA UK, IEEE, IMech

Assistant Lecturer at SLIIT | Senior Lecturer at Academy of Innovative Education

DEEP LEARNING & NEURAL NETWORKS

CERTIFICATE COURSE

12	4	1	12
WEEKS	INCLASS PROJECTS	FINAL PROJECT	ASSIGNMENTS/ TUTORIALS

Detailed Course Content

- https://sliai.lk/deep-learning-neural-networks/

Method of Conduct

- Face to Face Lectures

(Every Week)

- Interactive Videos
- 4 In class Practical
- Assignments/ Tutorials

(Every Week)

- 1 Final Group Project
- All the Materials are available @ https://github.com/aieml

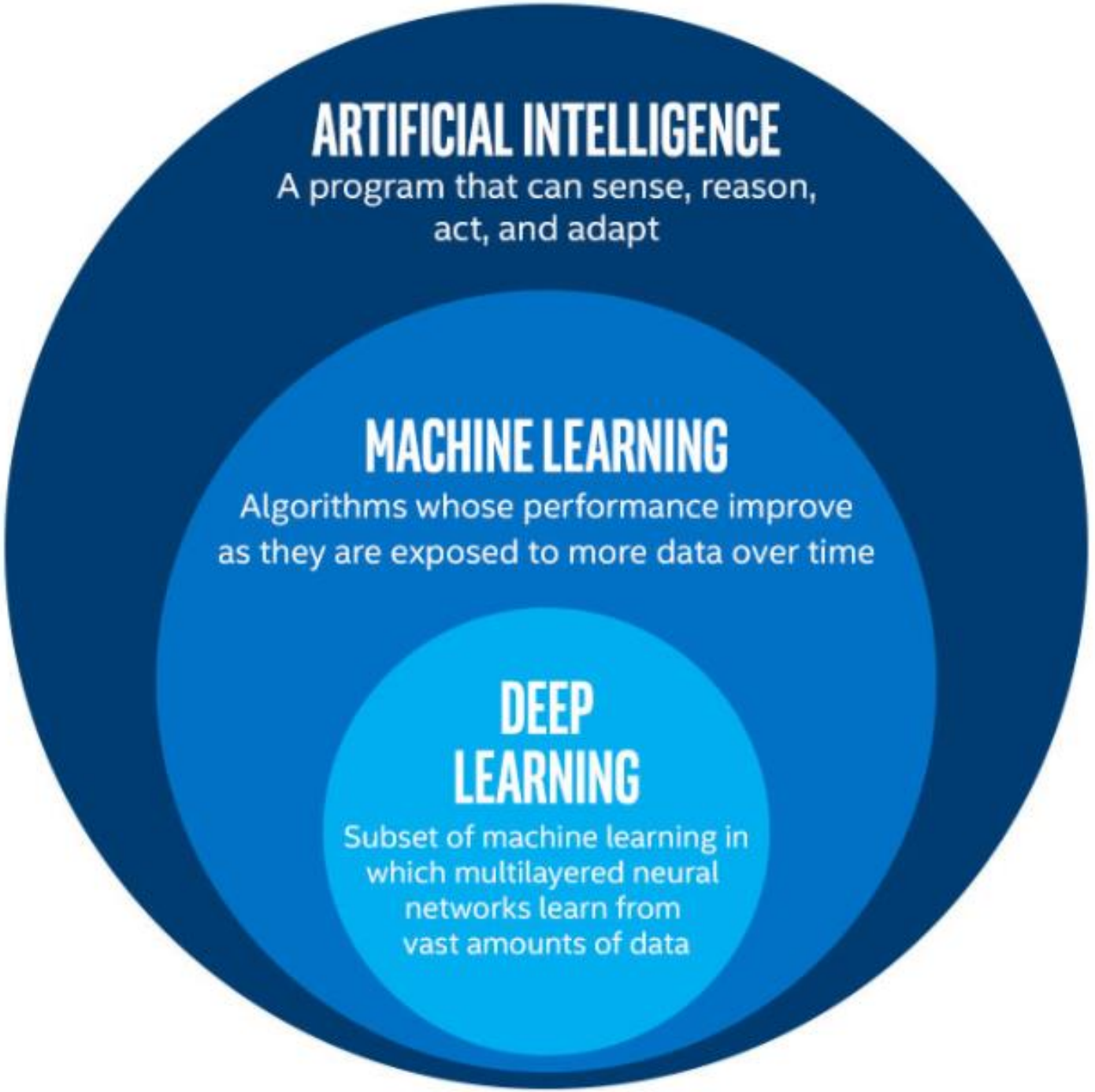
In class Projects

1. Training Object Detection API for Custom Objects (CNN)
2. Self Driving Car Simulator (CNN)
3. Stock Market Value Prediction (DQN)
4. Time Series Data Analysis (RNN)

Previously

- https://sliai.lk/machine-learning-image-processing/
- https://www.facebook.com/SLAIEtheORIGINAL/photos/pcb.2291235700954123/2291232597621100/?type=3&theater

AI vs ML vs DL


ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act, and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amounts of data

DEEP LEARNING

Allow the computers learn automatically without human intervention or assistance and adjust actions accordingly

Supervised DL

learned in the past to new data using labeled examples to predict future events

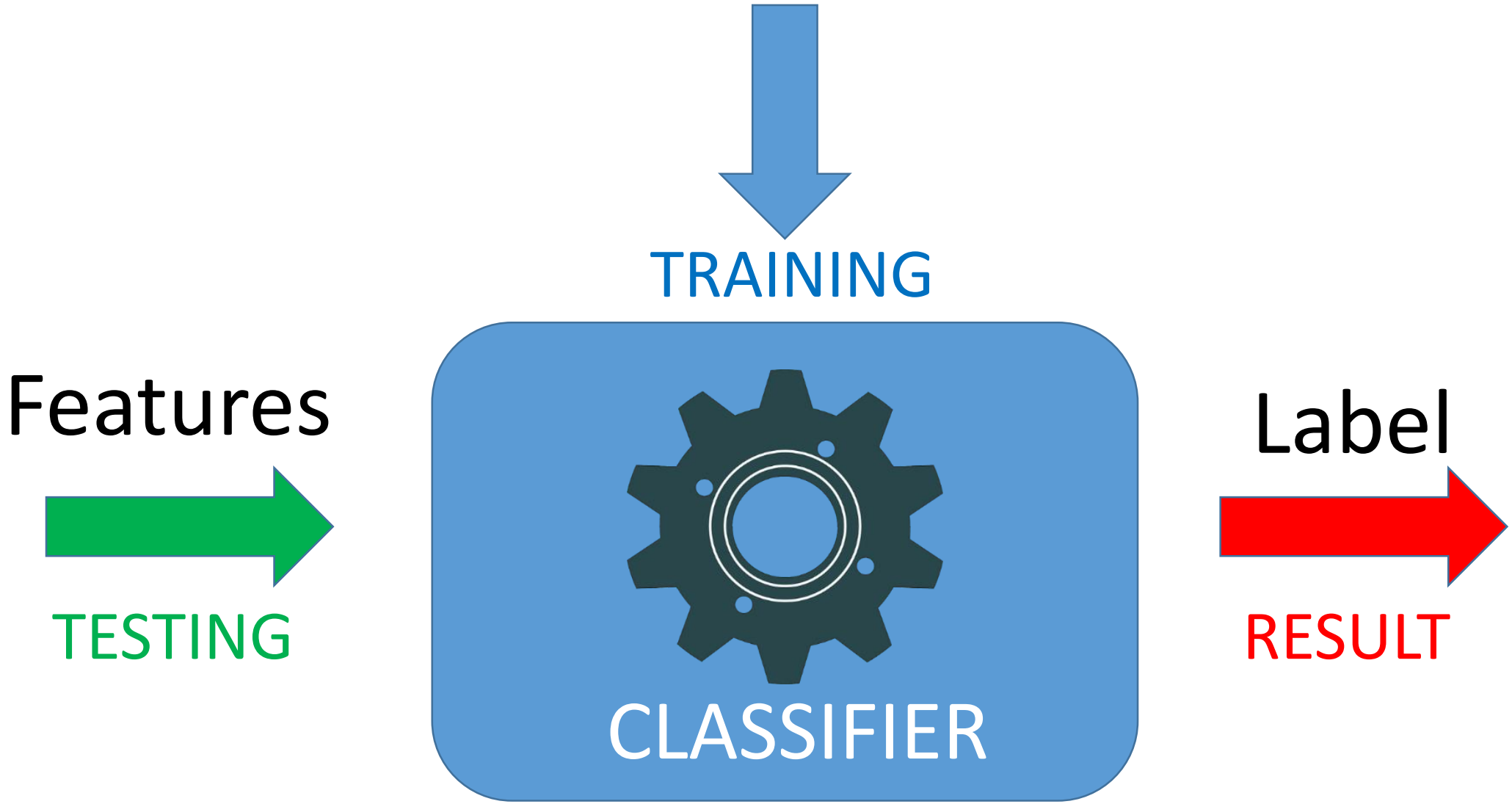
Unsupervised DL

Used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data

Reinforcement DL

Agent learns in a environment to achieve a long term goal by maximizing short term rewards

Features & Labels



(120,smooth), Apple
(140,Rough),Orange
(135,Smooth),Orange
(115,Smooth),Apple
(170,Rough),Orange
(165,Rough),Apple

TRAINING

(150,rough)

TESTING

CLASSIFIER

Apple/Orange

RESULT

Why You Should Learn ML

- It is the Future Industry
- Research Opportunities for Undergraduates / Post Graduates
- Data Scientist
- Big Data analysis
- AI Engineers
- AI based IOT and Mobile App Development
- Finance and Statistics Sector
- Stock Market Data Analysis, share value Prediction
- Automobile Industry
- Medical Industry

DEEP LEARNING

“In a neural network we don't tell the computer how to solve our problem. Instead, it learns from observational data, figuring out its own solution to the problem at hand.”

-In 2006 was the discovery of techniques for learning in so-called deep neural networks. These techniques are now known as deep learning. They've been developed further, and today deep neural networks and deep learning achieve outstanding performance on many important problems in computer vision, speech recognition, and natural language processing-

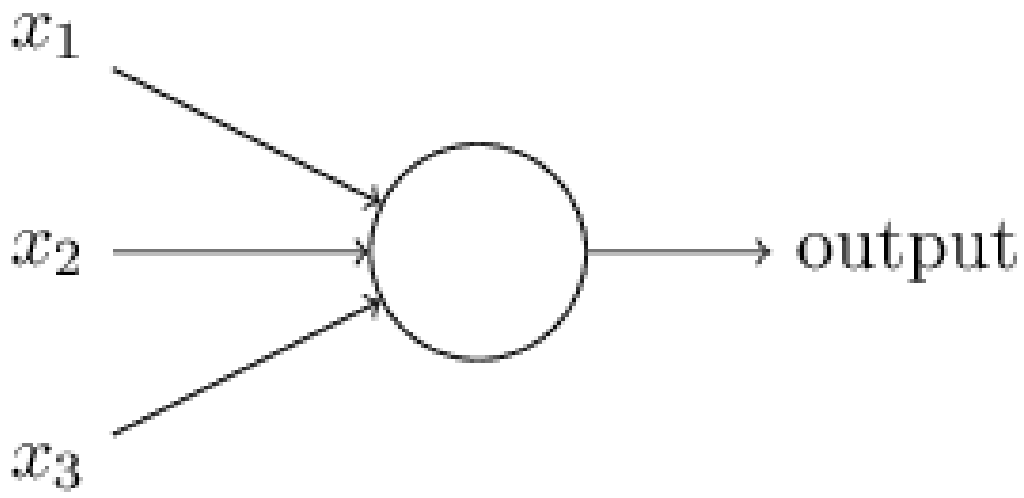
They use Deep Learning

Applications

- Face Recognition
- Image Classification
- Speech Recognition
- Text-to-speech Generation
- Handwriting Transcription
- Machine Translation
- Medical Diagnosis
- Self Driving Cars
- Digital Assistants
- Ads, search, social recommendations
- Games with Deep Reinforcement Learning

History In Brief (1)

- The idea of neural networks began unsurprisingly as a model of how neurons in the brain function.
  - 1943: Portrayed with a simple electrical circuit by neurophysiologist Warren McCulloch and mathematician Walter Pitt
  - 1950-1960: Perceptrons were developed by the scientist Frank Rosenblatt,



History In Brief (2)

- 1974-86: Backpropagation Algorithm, Recurrent NL
- 1989-98: Convolutional Neural Networks, Bi Directional RNN, Long Short Term Memory (LSTM), MNIST Data Set
- 2006: “Deep Learning” Concept
- 2009: ImageNet
- 2012: AlexNet, Dropout
- 2014: DeepFace
- 2016: AlphaGo
- 2017: AlphaGo Zero
- 2018: BERT

History In Brief (3)

- Mark 1 Perceptron: 1960
- Torch: 2002
- CUDA: 2007
- Theano: 2008
- Caffe: 2015
- Tensorflow 0.1: 2015
- PyTorch 0.1: 2017
- TensorFlow 1.0: 2017
- Tensorflow 2.0: 2019

SOFTWARE TOOLS

PYTHON

Programming Language

OPEN SOURCE LIBRARIES

JUPYTER NOTEBOOK & GOOGLE COLAB

Development Environments

WEEK 01 - MACHINE LEARNING

ESSENTIAL PYTHON & CONTRIBUTED MODULES

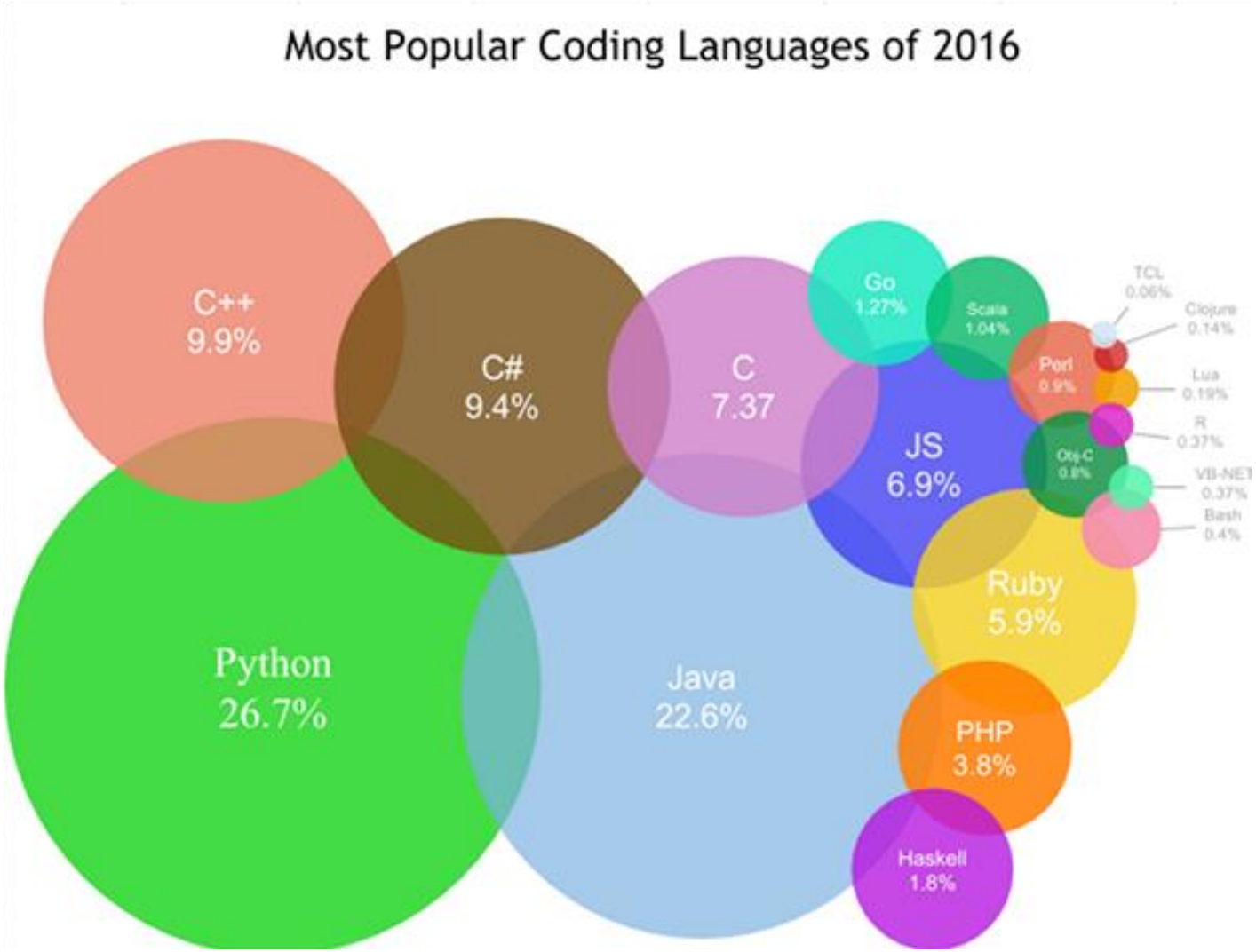
Thakshila Dasun
BSc. Hons in Mechanical Engineering
(Mechatronics Specialization)
CIMA, UK
Academy of Innovative Education

COMPULSORY FINAL PROJECT

- Everyone has to involve in a Group Projects
- Topics will be given in the 3rd Week
- The Final Grade of the Certificate will be decided by the performances

INTRODUCTION (1)

- Developed by Guido van Rossum in the late eighties and early nineties.
- Derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- HIGH LEVEL Scripting Language



SOURCE: http://blog.codeeval.com/codeevalblog/2016/2/2/most-popular-coding-languages-of-2016

INTRODUCTION (2)

- **Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it (PERL and PHP).
- **Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

INTRODUCTION (2)

- Supports functional and structured programming methods as well as OOP.
- can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Modes of Python Programming(1)

1. Interactive Mode Programming:
2. Script Mode Programming

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module, or other object.
- An identifier starts with a letter A to Z or a to z, or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers.
- Python is a case sensitive programming language.

Python Keywords

- The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

And	exec	Not
Assert	finally	or
Break	for	pass
Class	from	print
Continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Lines and Indentation

```
if True:
    print "True"
else:
  print "False"
```

```
if True:
    print "Answer"
    print "True"
else:
    print "Answer"
  print "False"
```

Comments in Python

- A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

Assigning Values to Variables

```
#!/usr/bin/python

counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print counter
print miles
print name
```

Multiple Assignment

```
a = b = c = 1
```

```
a,b,c = 1,2,"john"
```

Strings

- Treats single quotes the same as double quotes.
- Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

Escape Characters

- Following table is a list of escape or non-printable characters that can be represented with backslash notation.
- An escape character gets interpreted; in a single quoted as well as double quoted strings.

Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0.7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F

String Formatting Operator

- This operator is unique to strings and makes up for the pack of having functions from C's printf() family.

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

Built-in String Methods

See the PDF

- Capitalize
- Min, max
- Upper
- len

Different numerical types

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Arithmetic Operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Comparison operators are same as in C

Assignment Operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Mathematical Functions

Function	Returns (description)
abs(x)	The absolute value of x: the (positive) distance between x and zero.
ceil(x)	The ceiling of x: the smallest integer not less than x
cmp(x, y)	-1 if x < y, 0 if x == y, or 1 if x > y
exp(x)	The exponential of x: e^x
fabs(x)	The absolute value of x.
floor(x)	The floor of x: the largest integer not greater than x
log(x)	The natural logarithm of x, for x> 0
log10(x)	The base-10 logarithm of x for x> 0 .
max(x1, x2,...)	The largest of its arguments: the value closest to positive infinity
min(x1, x2,...)	The smallest of its arguments: the value closest to negative infinity
modf(x)	The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
pow(x, y)	The value of x**y.
round(x [,n])	x rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
sqrt(x)	The square root of x for x > 0

NOTE: (complex numbers), math library

Lists

- Most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets.
- Important thing about a list is that items in a list need not be of the same type.

Built-in List Functions & Methods I

SN	Function with Description
1	**cmp(list1, list2)** Compares elements of both lists.
2	**len(list)** Gives the total length of the list.
3	**max(list)** Returns item from the list with max value.
4	**min(list)** Returns item from the list with min value.
5	**list(seq)** Converts a tuple into list.

Built-in List Functions & Methods II

SN	Methods with Description
1	**list.append(obj)** Appends object obj to list
2	**list.count(obj)** Returns count of how many times obj occurs in list
3	**list.extend(seq)** Appends the contents of seq to list
4	**list.index(obj)** Returns the lowest index in list that obj appears
5	**list.insert(index, obj)** Inserts object obj into list at offset index
6	**list.pop(obj=list[-1])** Removes and returns last object or obj from list
7	**list.remove(obj)** Removes object obj from list
8	**list.reverse()** Reverses objects of list in place
9	**list.sort([func])** Sorts objects of list, use compare func if given

Random Number Functions

Function	Description
choice(seq)	A random item from a list, tuple, or string.
randrange ([start,] stop [,step])	A randomly selected element from range(start, stop, step)
random()	A random float r, such that 0 is less than or equal to r and r is less than 1
seed([x])	Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.
shuffle(lst)	Randomizes the items of a list in place. Returns None.
uniform(x, y)	A random float r, such that x is less than or equal to r and r is less than y

Tuple

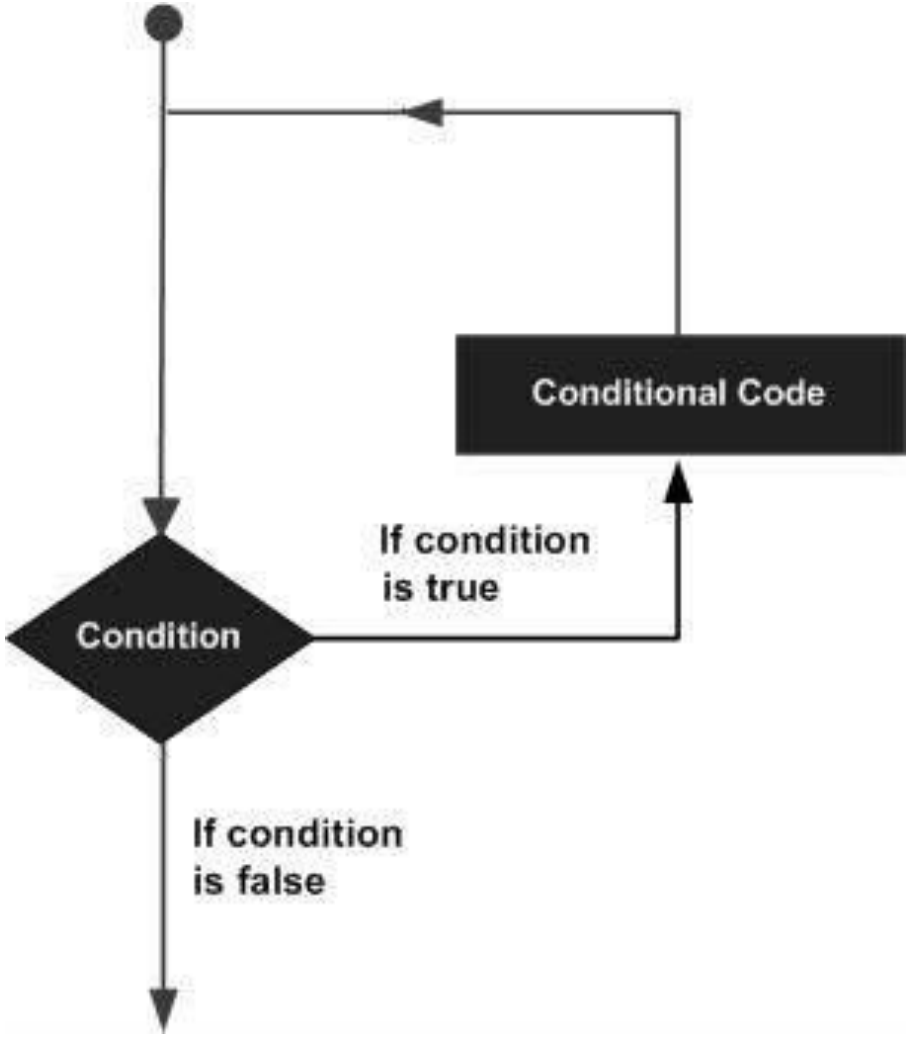
- A tuple is a sequence of immutable Python objects.
- Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Dictionary

- Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces.
- An empty dictionary without any items is written with just two curly braces, like this: {}

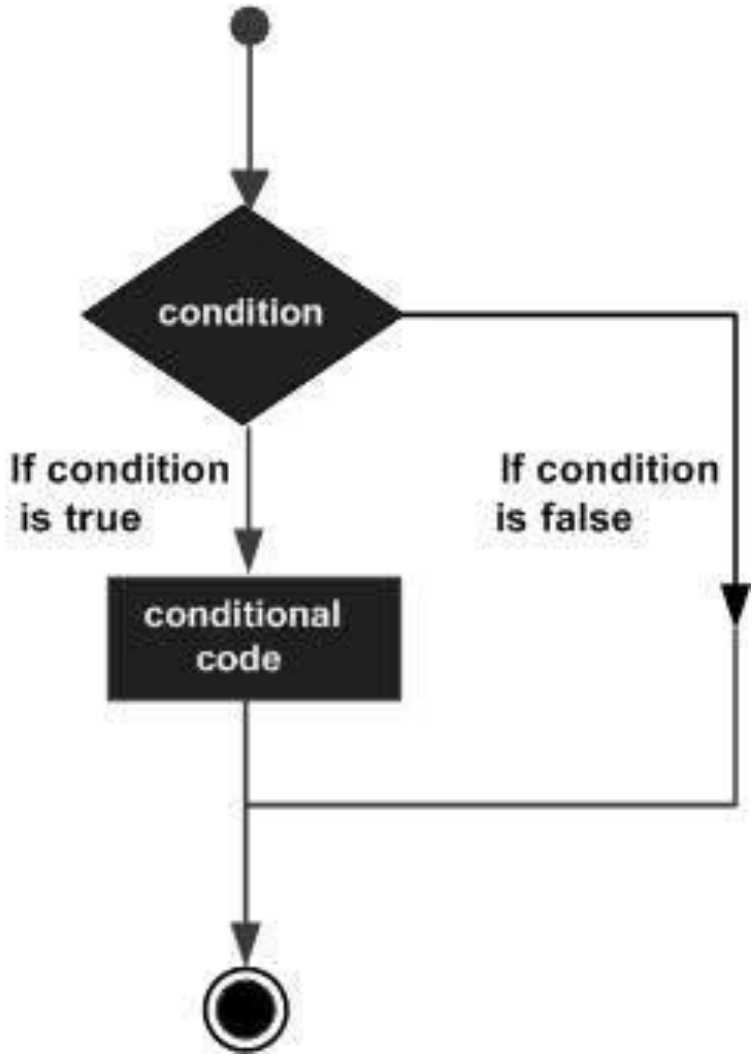
Loops

- A loop statement allows us to execute a statement or group of statements multiple times.



Decision Making

- Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome.
- You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

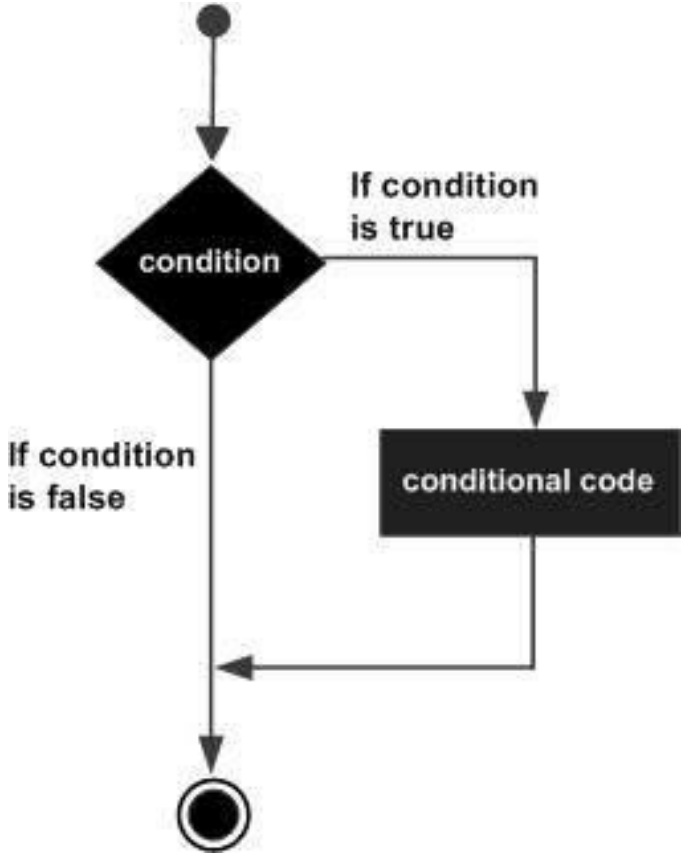


DECISION MAKING

- Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.
- Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome.
- Python assumes any **non-zero** and **non-null** values as TRUE, and if it is either **zero** or **null**, then it is assumed as FALSE value.

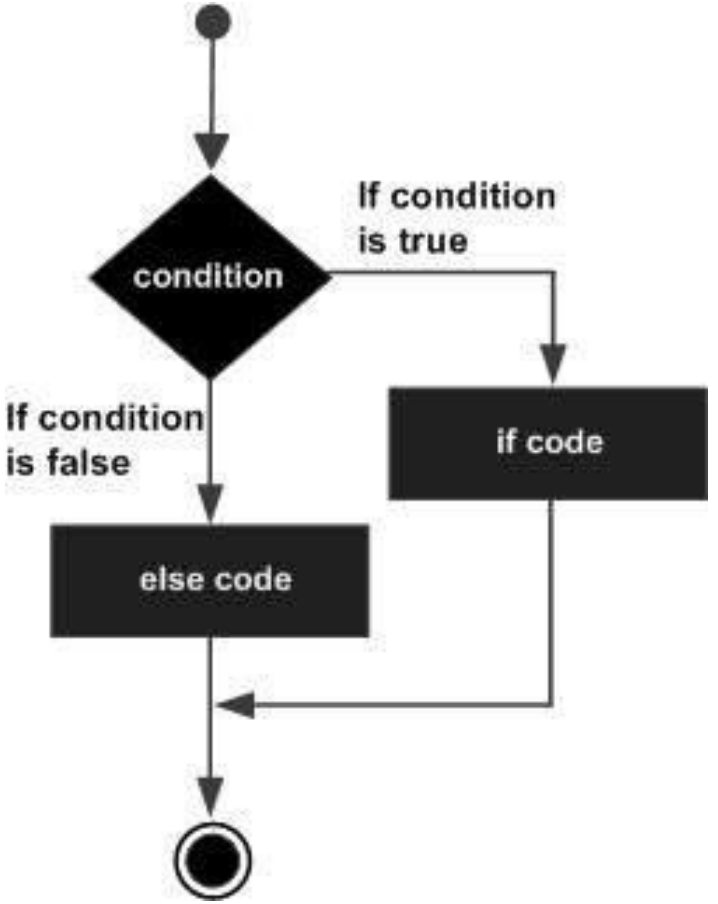
If STATEMENT

- Consists of a boolean expression followed by one or more statements.
- Contains a logical expression using which data is compared and a decision is made based on the result of the comparison.
- If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed.
- If boolean expression evaluates to FALSE, then the first set of code after the end of the if statement(s) is executed.



If-else STATEMENT

- Can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
- An **else** statement can be combined with an **if** statement.
- It contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.



NOTE: The *else* statement is an optional statement and there could be at most only one **else** statement following **if** .

NESTED if STATEMENTS (if-elif….-else)

- When you want to check for another condition after a condition resolves to true.
- In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

Built-in List Functions & Methods I

SN	Function with Description
1	**cmp(list1, list2)** Compares elements of both lists.
2	**len(list)** Gives the total length of the list.
3	**max(list)** Returns item from the list with max value.
4	**min(list)** Returns item from the list with min value.
5	**list(seq)** Converts a tuple into list.

Built-in List Functions & Methods II

SN	Methods with Description
1	**list.append(obj)** Appends object obj to list
2	**list.count(obj)** Returns count of how many times obj occurs in list
3	**list.extend(seq)** Appends the contents of seq to list
4	**list.index(obj)** Returns the lowest index in list that obj appears
5	**list.insert(index, obj)** Inserts object obj into list at offset index
6	**list.pop(obj=list[-1])** Removes and returns last object or obj from list
7	**list.remove(obj)** Removes object obj from list
8	**list.reverse()** Reverses objects of list in place
9	**list.sort([func])** Sorts objects of list, use compare func if given

Random Number Functions (*random* module)

Function	Description
choice(seq)	A random item from a list, tuple, or string.
randrange ([start,] stop [,step])	A randomly selected element from range(start, stop, step)
random()	A random float r, such that 0 is less than or equal to r and r is less than 1
seed([x])	Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.
shuffle(lst)	Randomizes the items of a list in place. Returns None.
uniform(x, y)	A random float r, such that x is less than or equal to r and r is less than y

FUNCTIONS (1)

- A block of organized, reusable code that is used to perform a single, related action.
- Provide better modularity for your application and a high degree of code reusing.
- Defining a Function
  - begin with the keyword **def** followed by the function name and parentheses (( )).
  - Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
  - The code block within every function starts with a colon (:) and is indented.
  - The statement return [expression] exits a function, optionally passing back an expression to the caller.
- Calling a Function
  - Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

NOTE: A return statement with no arguments is the same as return None.

FUNCTIONS (2)

- Pass by reference vs value
  - All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

FUNCTIONS (3)

- Function Arguments

    1. Required arguments
    2. Keyword arguments
    3. Default arguments
    4. Variable-length arguments

Required arguments

- The arguments passed to a function in correct positional order.
- The number of arguments in the function call should match exactly with the function definition.

Keyword arguments

- Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
- This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

Default arguments

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

Variable-length arguments

- You may need to process a function for more arguments than you specified while defining the function.
- These arguments are called *variable-length* arguments and are not named in the function definition, unlike required and default arguments.
- An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call

The *Anonymous* Functions (1)

- These functions are called anonymous because they are not declared in the standard manner by using the *def* keyword.
- You can use the *lambda* keyword to create small anonymous functions.

The *Anonymous* Functions (2)

- Lambda
  - Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
  - An anonymous function cannot be a direct call to print because lambda requires an expression
  - Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
  - Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C or C++,

The *return* Statement

- The statement return [expression] exits a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as return None.
- Multiple values can be returned, and it will be saved in a tuple

Scope of Variables

- All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.
- The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python.

Global vs. Local variables

- Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.
- This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions.
- When you call a function, the variables declared inside it are brought into scope.