POLYTECHNIC UNIVERSITY OF THE PHILIPPINES
COLLEGE OF COMPUTER AND INFORMATION SCIENCES
DEPARTMENT OF COMPUTER SCIENCE | DEPARTMENT OF INFORMATION TECHNOLOGY

Integrative Programming and Technologies 1

INSTRUCTIONAL MATERIAL FOR STUDENTS

Compiled by:

ELIAS A. AUSTRIA
ALETA C. FABREGAS
RACHEL A. NAYRE

TABLE OF CONTENTS

Lesson 1 – Introduction to Python

Overview

This lesson covers an overview of Python language. It also instruct on how the Python compiler will be installed and set up using different platforms.

Objectives

At the end of the lesson, the student will be able to:

- Understand the overview of Python language
- Experience to install and set up the language using different operating system

General Course Materials

1. Python Tutorials for Beginners https://www.guru99.com/python-tutorials.html
2. **Python 3 https://www.tutorialspoint.com/python3/python_tutorial.pdf**
3. **Python Full Course - Learn Python in 12 Hours | Python Tutorial For Beginners | Edureka https://www.youtube.com/watch?v=WGJJIrtnfpk**

Overview of Python Language

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This **tutorial** gives enough understanding on **Python programming** language.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain.

Advantages of learning Python:

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a 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.

Characteristics of Python Programming

- It supports functional and structured programming methods as well as OOP.
- It 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.

Features of Python

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

Local Environment Setup

Open a terminal window and type "python" to find out if it is already installed and which version is installed.

- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, etc.)
- Win 9x/NT/2000
- Macintosh (Intel, PPC, 68K)
- OS/2
- DOS (multiple versions)
- PalmOS
- Nokia mobile phones

- Windows CE
- Acorn/RISC OS
- BeOS
- Amiga
- VMS/OpenVMS
- QNX
- VxWorks
- Psion
- Python has also been ported to the Java and .NET virtual machines

Getting Started to Python

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python https://www.python.org/

You can download Python documentation from https://www.python.org/doc/. The documentation is available in HTML, PDF, and PostScript formats.

Installing Python

Python distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install Python.
If the binary code for your platform is not available, you need a C compiler to compile the source code manually. Compiling the source code offers more flexibility in terms of choice of features that you require in your installation.
Here is a quick overview of installing Python on various platforms –

Unix and Linux Installation

Here are the simple steps to install Python on Unix/Linux machine.

- Open a Web browser and go to https://www.python.org/downloads/.
- Follow the link to download zipped source code available for Unix/Linux.
- Download and extract files.
- Editing the *Modules/Setup* file if you want to customize some options.
- run ./configure script
- make
- make install

This installs Python at standard location */usr/local/bin* and its libraries at */usr/local/lib/pythonXX* where XX is the version of Python.

Windows Installation

Here are the steps to install Python on Windows machine.

- Open a Web browser and go to https://www.python.org/downloads/.
- Follow the link for the Windows installer *python-XYZ.msi* file where XYZ is the version you need to install.
- To use this installer *python-XYZ.msi*, the Windows system must support Microsoft Installer 2.0. Save the installer file to your local machine and then run it to find out if your machine supports MSI.
- Run the downloaded file. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the install is finished, and you are done.

Macintosh Installation

Recent Macs come with Python installed, but it may be several years out of date. See http://www.python.org/download/mac/ for instructions on getting the current version along with extra tools to support development on the Mac. For older Mac OS's before Mac OS X 10.3 (released in 2003), MacPython is available.

Jack Jansen maintains it and you can have full access to the entire documentation at his website − http://www.cwi.nl/~jack/macpython.html. You can find complete installation details for Mac OS installation.

Setting up PATH

Programs and other executable files can be in many directories, so operating systems provide a search path that lists the directories that the OS searches for executable.

The path is stored in an environment variable, which is a named string maintained by the operating system. This variable contains information available to the command shell and other programs.

The **path** variable is named as PATH in Unix or Path in Windows (Unix is case sensitive; Windows is not).

In Mac OS, the installer handles the path details. To invoke the Python interpreter from any particular directory, you must add the Python directory to your path.

Setting path at Unix/Linux

To add the Python directory to the path for a particular session in Unix −

- **In the csh shell** − type setenv PATH "$PATH:/usr/local/bin/python" and press Enter.
- **In the bash shell (Linux)** − type export PATH="$PATH:/usr/local/bin/python" and press Enter.
- **In the sh or ksh shell** − type PATH="$PATH:/usr/local/bin/python" and press Enter.
- **Note** − /usr/local/bin/python is the path of the Python directory

Setting path at Windows

To add the Python directory to the path for a particular session in Windows −

At the command prompt − type path %path%;C:\Python and press Enter.

Note − C:\Python is the path of the Python directory

Assessment/Activities

1. Install python platform in your laptop

Lesson 2 – Basic Concepts and Structure

Overview

This lesson includes the basic components of Python and its structure. It will help the student to understand more about language and to compare it to other programming language/s they have learned.

Objectives

At the end of the lesson, the students will:

- Learn the different components and structure of Python
- Further understand the language
- Create and execute the program written in the language

General Course Materials

1. Python Tutorials for Beginners https://www.guru99.com/python-tutorials.html
2. **Python 3 https://www.tutorialspoint.com/python3/python_tutorial.pdf**
3. **Python Full Course - Learn Python in 12 Hours | Python Tutorial For Beginners | Edureka https://www.youtube.com/watch?v=WGJJIrtnfpk**

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. Thus, **Manpower** and **manpower** are two different identifiers in Python.

Naming conventions of Python identifiers:

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.

Statements contained within the [], {}, or () brackets do not need to use the line continuation character.

Quotation

Python accepts single ('), double (") and triple ('" or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines.

Comments

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.

Multiple Statements on a Single Line

The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block.

Multiple Statement Groups as Suites

A group of individual statements, which make a single code block are called **suites** in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite.

Reserved Words

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	except	is	while
assert	exec	lambda	with
break	finally	not	yield
class	for	or	
continue	from	pass	
def	global	print	
del	if	raise	
elif	import	return	
else	in	try	

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

Python allows you to assign a single value to several variables simultaneously.

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

a. Numbers

Number data types store numeric values. They are immutable data types, means that changing the value of a number data type results in a newly allocated object.

Python supports four different numerical types −

- **int (signed integers)** − They are often called just integers or ints, are positive or negative whole numbers with no decimal point.
- **long (long integers)** − Also called longs, they are integers of unlimited size, written like integers and followed by an uppercase or lowercase L.
- **float (floating point real values)** − Also called floats, they represent real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 (2.5e2 = 2.5 x 10^2 = 250).
- **complex (complex numbers)** − are of the form a + bJ, where a and b are floats and J (or j) represents the square root of -1 (which is an imaginary number). The real part of the number is a, and the imaginary part is b. Complex numbers are not used much in Python programming.

b. **Strings**

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

Accessing Values in Strings

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

Updating Strings

You can "update" an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether.

String Special Operators

Assume variable ‘a’ is “Hello” and variable ‘b’ is “Phyton”:

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1
r/R	Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.	print r'\n' prints \n and print R'\n'prints \n
%	Format - Performs String formatting	See at next section

String Formatting Operators

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

c. List

- Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

d. Tuple

- A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists.

e. Dictionary

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Function & Description
int(x [,base]) Converts x to an integer. base specifies the base if x is a string.
long(x [,base]) Converts x to a long integer. base specifies the base if x is a string.

float(x) Converts x to a floating-point number.
complex(real [,imag]) Creates a complex number.
str(x) Converts object x to a string representation.
repr(x) Converts object x to an expression string.
eval(str) Evaluates a string and returns an object.
tuple(s) Converts s to a tuple.
list(s) Converts s to a list.
set(s) Converts s to a set.
dict(d) Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s) Converts s to a frozen set.
chr(x) Converts an integer to a character.
unichr(x) Converts an integer to a Unicode character.
ord(x) Converts a single character to its integer value.
hex(x) Converts an integer to a hexadecimal string.
oct(x) Converts an integer to an octal string.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators (+, –, *, /, %, **, //)

- Comparison (Relational) Operators (<, <=, > >=, ==, !=)
- Assignment Operators (=, +=, –=, *=, /=, %=. **=, //=)
- Logical Operators (not, and, or)
- Bitwise Operators (&, |, ^, ~, <<, >>)
- Membership Operators (in, not in)
- Identity Operators (is, is not)

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. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

if statements An **if statement** consists of a boolean expression followed by one or more statements.
if...else statements An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is FALSE.
nested if statements You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

Loop Statements

while loop Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops You can use one or more loop inside any another while, for or do..while loop.

Loop Control Statements

break statement Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

pass statement
The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Escape Characters

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

Assessment /Activities

1. Write a Python program to print the following string in a specific format (see the output).

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.

Twinkle, twinkle, little star,
How I wonder what you are

2. Write a Python program to display the current date and time.
   Sample Output :
   Current date and time :
   2014-07-05 14:34:14
3. Write a Python program which accepts the radius of a circle from the user and compute the area.
   Sample Output :
   r = 1.1
   Area = 3.8013271108436504
4. Write a Python program which accepts the user's first and last name and print them in reverse order with a space between them.
5. Write a Python program to find whether a given number (accept from the user) is even or odd, print out an appropriate message to the user.
6. Write a Python program that will accept the base and height of a triangle and compute the area.

Lesson 3 – List, Tuple and Dictionary

Overview

This lesson introduces the List, Tuple and Dictionary as applied to different examples for further learning. Different methods and functions are discussed with list, tuple and dictionary.

Objectives

At the end of the lesson, the students will:

- Learn the basic concepts of List, Tuple and Dictionary
- Further understand list, tuple and dictionary.
- Create and execute the python program with list, tuple and dictionary applications.

General Course Materials

1. Python Tutorials for Beginners https://www.guru99.com/python-tutorials.html
2. **Python 3 https://www.tutorialspoint.com/python3/python_tutorial.pdf**
3. **Python Full Course - Learn Python in 12 Hours | Python Tutorial For Beginners | Edureka https://www.youtube.com/watch?v=WGJJIrtnfpk**

List

The most basic data structure in Python is the **sequence**. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.

Python has six built-in types of sequences, but the most common ones are lists and tuples, which we would see in this tutorial.

There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements.

The list is a 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.

Creating a list is as simple as putting different comma-separated values between square brackets. For examples:

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

Accessing Values in Lists

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.

Updating Lists

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method.

Delete List Elements

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.

Basic List Operations

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Built-in List Functions & Methods

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

Methods with Description
list.append(obj) Appends object obj to list
list.count(obj) Returns count of how many times obj occurs in list
list.extend(seq) Appends the contents of seq to list
list.index(obj) Returns the lowest index in list that obj appears

list.insert(index, obj) Inserts object obj into list at offset index
list.pop(obj=list[-1]) Removes and returns last object or obj from list
list.remove(obj) Removes object obj from list
list.reverse() Reverses objects of list in place
list.sort([func]) Sorts objects of list, use compare func if given

Tuple

A tuple is a collection of objects which ordered and immutable. 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.
Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also.

Accessing Values in Tuples
To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.

Updating Tuples
Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples.

Delete Tuple Elements
Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement.

Basic Tuples Operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership

for x in (1, 2, 3): print x,	1 2 3	Iteration

Built-in Tuple Functions

Function with Description
cmp(tuple1, tuple2) Compares elements of both tuples.
len(tuple) Gives the total length of the tuple.
max(tuple) Returns item from the tuple with max value.
min(tuple) Returns item from the tuple with min value.
tuple(seq) Converts a list into tuple.

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: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.

Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry

Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.
To explicitly remove an entire dictionary, just use the **del** statement.

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.
There are two important points to remember about dictionary keys −
(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

Built-in Dictionary Functions & Methods

Function with Description
cmp(dict1, dict2) Compares elements of both dict.
len(dict) Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
str(dict) Produces a printable string representation of a dictionary
type(variable) Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.
Methods with Description
dict.clear() Removes all elements of dictionary dict
dict.copy() Returns a shallow copy of dictionary dict
dict.fromkeys() Create a new dictionary with keys from seq and values set to value.
dict.get(key, default=None) For key key, returns value or default if key not in dictionary
dict.has_key(key) Returns true if key in dictionary dict, false otherwise
dict.items() Returns a list of dict's (key, value) tuple pairs
dict.keys() Returns list of dictionary dict's keys
dict.setdefault(key, default=None) Similar to get(), but will set dict[key]=default if key is not already in dict
dict.update(dict2) Adds dictionary dict2's key-values pairs to dict
dict.values() Returns list of dictionary dict's values

Assessment/Activities

1. Printing numbers 1 to 10 on the screen using a list to store the numbers:
2. Create a list with five objects and showing on the screen
3. Create a list with five objects and showing on the screen
4. Create a tuple with the following elements: 1, 2, 2, 3, 4, 4, 4, 5, and then use the tuple object's count function to check how many times the number for appears in the tuple.

Lesson 4 – Function

Overview

This lesson provides a better understanding of program written with function code. The purpose of function to make the program more efficient and effective program coding technique. It is very useful for error detection and will not affect the execution of the entire program It also discusses the different built-in functions.

Objectives

At the end of the lesson, the students will:

- Understand the use of function in the program
- Recognize the variable scoping
- Code a program with function
- Apply the built-in functions in the program code

General Course Materials

1. Python Tutorials for Beginners https://www.guru99.com/python-tutorials.html
2. Python 3 https://www.tutorialspoint.com/python3/python_tutorial.pdf
3. Python Full Course - Learn Python in 12 Hours | Python Tutorial For Beginners | Edureka https://www.youtube.com/watch?v=WGJJIrtnfpk

Function

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks 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 first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- 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. A return statement with no arguments is the same as return None.

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.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

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.

Function Arguments

You can call a function by using the following types of formal arguments –

- **Required arguments**
  Required arguments are the arguments passed to a function in correct positional order. Here, 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

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.

- 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++, whose purpose is by passing function stack allocation during invocation for performance reasons.

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.

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 variables
- Local variables

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.

Built-in Functions

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

Random Number Functions

Random numbers are used for games, simulations, testing, security, and privacy applications.

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

String Functions

Methods with Description
capitalize() Capitalizes first letter of string
center(width, fillchar) Returns a space-padded string with the original string centered to a total of width columns.
count(str, beg= 0,end=len(string)) Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
decode(encoding='UTF-8',errors='strict') Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.
encode(encoding='UTF-8',errors='strict') Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.
endswith(suffix, beg=0, end=len(string)) Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
expandtabs(tabsize=8) Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.
find(str, beg=0 end=len(string)) Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.
index(str, beg=0, end=len(string)) Same as find(), but raises an exception if str not found.
isalnum() Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
isalpha() Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
isdigit() Returns true if string contains only digits and false otherwise.
islower() Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
isnumeric()

Returns true if a unicode string contains only numeric characters and false otherwise.
isspace() Returns true if string contains only whitespace characters and false otherwise.
istitle() Returns true if string is properly "titlecased" and false otherwise.
isupper() Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
join(seq) Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
len(string) Returns the length of the string
ljust(width[, fillchar]) Returns a space-padded string with the original string left-justified to a total of width columns.
lower() Converts all uppercase letters in string to lowercase.
lstrip() Removes all leading whitespace in string.
maketrans() Returns a translation table to be used in translate function.
max(str) Returns the max alphabetical character from the string str.
min(str) Returns the min alphabetical character from the string str.
replace(old, new [, max]) Replaces all occurrences of old in string with new or at most max occurrences if max given.
rfind(str, beg=0,end=len(string)) Same as find(), but search backwards in string.
rindex(str, beg=0, end=len(string)) Same as index(), but search backwards in string.
rjust(width,[, fillchar]) Returns a space-padded string with the original string right-justified to a total of width columns.
rstrip() Removes all trailing whitespace of string.

split(str="", num=string.count(str)) Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
splitlines(num=string.count('\n')) Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
startswith(str, beg=0,end=len(string)) Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.
strip([chars]) Performs both lstrip() and rstrip() on string.
swapcase() Inverts case for all letters in string.
title() Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.
translate(table, deletechars="") Translates string according to translation table str(256 chars), removing those in the del string.
upper() Converts lowercase letters in string to uppercase.
zfill (width) Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).
isdecimal() Returns true if a unicode string contains only decimal characters and false otherwise.

Assessment/Activities

1. What is the output of the following function call

```
def fun1(name, age=20):
    print(name, age)

fun1('Emma', 25)
```

- Emma 25
- Emma 20

2. Select which true for Python function

- A function is a code block that only executes when it is called.

- Python function always returns a value.
- A function only executes when it is called and we can reuse it in a program
- Python doesn't support nested function

3. Select which is true for Python function

- A Python function can return only a single value
- A function can take an unlimited number of arguments.
- A Python function can return multiple values
- Python function doesn't return anything unless and until you add a return statement

4. Given the following function `fun1()` Please **select all the correct function calls**

```
def fun1(name, age):
    print(name, age)
```

1. `fun1("Emma", age=23)`
2. `fun1(age =23, name="Emma")`

- fun1(name="Emma", 23)
- fun1(age =23, "Emma")

5. Choose the correct function declaration of `fun1()` so that we can execute the following function call successfully

```
fun1(25, 75, 55)
fun1(10, 20)
```

- def fun1(**kwargs)
- No, it is not possible in Python
- def fun1(args*)
- def fun1(*data)

6. What is the output of the following code

```
def outerFun(a, b):
    def innerFun(c, d):
        return c + d
    return innerFun(a, b)

res = outerFun(5, 10)
```

```
print(res)
```

- 15
- Syntax Error
- (5, 10)

7. Python function always returns a value

- False
- True

8. What is the output of the `add()` function call

```
def add(a, b):
    return a+5, b+5

result = add(3, 2)
print(result)
```

- 15
- 8
- (8, 7)
- Syntax Error

9. What is the output of the following `displayPerson()` function call

```
def displayPerson(*args):
    for i in args:
        print(i)

displayPerson(name="Emma", age="25")
```

- TypeError
- Emma

25

- name

age

10. What is the output of the following `display()` function call

```
def display(**kwargs):
    for i in kwargs:
        print(i)
display(emp="Kelly", salary=9000)
```

- TypeError

- Kelly
9000
- ('emp', 'Kelly')
('salary', 9000)
- emp
salary

11. What is the output of the following function call

```
def outerFun(a, b):
    def innerFun(c, d):
        return c + d
    return innerFun(a, b)
    return a

result = outerFun(5, 10)
print(result)
```

- 5
- 15
- (15, 5)
- Syntax Error

12. What is the output of the following function call

```
def fun1(num):
    return num + 25

fun1(5)
print(num)
```

- 25
- 5
- NameError

Lesson 5 – Modules

Overview

This lessons will teach the student in writing a modules in Python language. Module let you logically systematize and easier to understand the program code.

Objectives

At the end of lesson, the students will have a knowledge in:

- Organizing the program code using the module
- Designing a program by applying module in the program code
- Organize modules into package

General Course Materials

- Python Tutorials for Beginners https://www.guru99.com/python-tutorials.html
- Python 3 https://www.tutorialspoint.com/python3/python_tutorial.pdf
- Python Full Course - Learn Python in 12 Hours | Python Tutorial For Beginners | Edureka https://www.youtube.com/watch?v=WGJJIrtnfpk

Module Definition

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

The *import* Statement

You can use any Python source file as a module by executing an import statement in some other Python source file.

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module.

A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur.

The *from...import* Statement

Python's *from* statement lets you import specific attributes from a module into the current namespace.

This provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly.

Locating Modules

When you import a module, the Python interpreter searches for the module in the following sequences:

- The current directory.
- If the module is not found, Python then searches each directory in the shell variable PYTHONPATH.
- If all else fails, Python checks the default path. On UNIX, this default path is normally /usr/local/lib/python/.

The module search path is stored in the system module sys as the **sys.path** variable. The sys.path variable contains the current directory, PYTHONPATH, and the installation-dependent default.

The *PYTHONPATH* Variable

The PYTHONPATH is an environment variable, consisting of a list of directories. The syntax of PYTHONPATH is the same as that of the shell variable PATH.

Here is a typical PYTHONPATH from a Windows system:

```
set PYTHONPATH = c:\python20\lib;
```

And here is a typical PYTHONPATH from a UNIX system:

```
set PYTHONPATH = /usr/local/lib/python
```

Namespaces and Scoping

Variables are names (identifiers) that map to objects. A *namespace* is a dictionary of variable names (keys) and their corresponding objects (values).

A Python statement can access variables in a *local namespace* and in the *global namespace*. If a local and a global variable have the same name, the local variable shadows the global variable.

Each function has its own local namespace. Class methods follow the same scoping rule as ordinary functions.

Python makes educated guesses on whether variables are local or global. It assumes that any variable assigned a value in a function is local.

Therefore, in order to assign a value to a global variable within a function, you must first use the global statement.

The statement *global VarName* tells Python that VarName is a global variable. Python stops searching the local namespace for the variable.

The dir( ) Function

The dir() built-in function returns a sorted list of strings containing the names defined by a module.

The list contains the names of all the modules, variables and functions that are defined in a module.

The *globals()* and *locals()* Functions

The *globals()* and *locals()* functions can be used to return the names in the global and local namespaces depending on the location from where they are called.

If locals() is called from within a function, it will return all the names that can be accessed locally from that function.

If globals() is called from within a function, it will return all the names that can be accessed globally from that function.

The return type of both these functions is dictionary. Therefore, names can be extracted using the keys() function.

The *reload()* Function

When the module is imported into a script, the code in the top-level portion of a module is executed only once.

Therefore, if you want to reexecute the top-level code in a module, you can use the *reload()* function. The reload() function imports a previously imported module again.

Packages in Python

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub-subpackages, and so on.

Assessment/Activities:

1. Which of these definitions correctly describes a module?
   a) Denoted by triple quotes for providing the specification of certain program elements
   b) Design and implementation of specific functionality to be incorporated into a program
   c) Defines the specification of how it is to be used
   d) Any program that reuses code
2. 2. Which of the following is not an advantage of using modules?
   a) Provides a means of reuse of program code
   b) Provides a means of dividing up tasks
   c) Provides a means of reducing the size of the program
   d) Provides a means of testing individual parts of the program
3. 3. Program code making use of a given module is called a _____ of the module.
   a) Client
   b) Docstring

c) Interface
d) Modularity

4. _____ is a string literal denoted by triple quotes for providing the specifications of certain program elements.
   a) Interface
   b) Modularity
   c) Client
   d) Docstring
5. Which of the following is true about top-down design process?
   a) The details of a program design are addressed before the overall design
   b) Only the details of the program are addressed
   c) The overall design of the program is addressed before the details
   d) Only the design of the program is addressed

6. What will be the output of the following Python code?

```
#mod1
def change(a):
    b=[x*2 for x in a]
    print(b)
#mod2
def change(a):
    b=[x*x for x in a]
    print(b)
from mod1 import change
from mod2 import change
#main
s=[1,2,3]
change(s)
```

a) [2,4,6]
b) [1,4,9]
c)

[2,4,6]
[1,4,9]

7. Which of the following is not a valid namespace?
   a) Global namespace
   b) Public namespace
   c) Built-in namespace
   d) Local namespace

Lesson 6 – Files

Objectives

At the end of lesson, the students will have a knowledge in:

- Attempt the File Manipulation tasks
- Demonstrate an understanding of File Manipulation
- Develop manipulation skills in Python

General Course Materials

1. Python Tutorials for Beginners https://www.guru99.com/python-tutorials.html
2. Python 3 https://www.tutorialspoint.com/python3/python_tutorial.pdf
3. Python Full Course - Learn Python in 12 Hours | Python Tutorial For Beginners | Edureka https://www.youtube.com/watch?v=WGJJIrtnfpk

Definition of File

A file is a collection of data stored in one unit, identified by a filename. It can be a document, picture, audio or video stream, data library, application, or other collection of data. The following is a brief description of each file type.

Documents include text files, such as a Word documents, RTF (Rich Text Format) documents, PDFs, Web pages, and others. Pictures include JPEGs, GIFs, BMPs, and layered image files, such as Photoshop documents (PSDs). Audio files include MP3s, AACs, WAVs, AIFs, and several others. Video files can be encoded in MPEG, MOV, WMV, or DV formats, just to name a few.

Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.

The *open* Function

Before you can read or write a file, you have to open it using Python's built-in *open()* function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```

Here are parameter details −

- **file_name** − The file_name argument is a string value that contains the name of the file that you want to access.

- **access_mode** − The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- **buffering** − If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

The *file* Object Attributes

Once a file is opened and you have one *file* object, you can get various information related to that file.
Here is a list of all attributes related to file object −

Attribute & Description
file.closed Returns true if file is closed, false otherwise.
file.mode Returns access mode with which file was opened.
file.name Returns name of the file.
file.softspace Returns false if space explicitly required with print, true otherwise.

The *close()* Method

The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Reading and Writing Files

The *file* object provides a set of access methods to make our lives easier. We would see how to use *read()* and *write()* methods to read and write files.

The *write()* Method

The *write()* method writes any string to an open file. It is important to note that Python have binary data and not just text. The write() method does not add a newline character ('\n') to the end of the string.

The *read()* Method

ethod reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

File Positions

The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The *seek(offset[, from])* method changes the current file position. The *offset* argument indicates the number of bytes to be moved. The *from* argument specifies the reference position from where the bytes are to be moved.

If *from* is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Renaming and Deleting Files

Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.
To use this module you need to import it first and then you can call any related functions.

The rename() Method

The *rename()* method takes two arguments, the current filename and the new filename.

The *remove()* Method

You can use the *remove()* method to delete files by supplying the name of the file to be deleted as the argument.

Directories in Python

All files are contained within various directories, and Python has no problem handling these too. The **os** module has several methods that help you create, remove, and change directories.

The *mkdir()* Method

You can use the *mkdir()* method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

The *chdir()* Method

You can use the *chdir()* method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.

The *getcwd()* Method

The *getcwd()* method displays the current working directory.

The *rmdir()* Method

The *rmdir()* method deletes the directory, which is passed as an argument in the method. Before removing a directory, all the contents in it should be removed.

File & Directory Related Methods

There are three important sources, which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows −

- File Object Methods: The *file* object provides functions to manipulate files.
- OS Object Methods: This provides methods to process files as well as directories.

Sample Programs in File Handling:

A.

```
#write some test data to a file
#1 Run the program

def writeFile():
    #open a file called 'temp.txt' in write mode
    file = open("temp.txt","w")

    #repeat 10 times
    for number in range(1,11):

        #create a string to write to the file
        line = "This is test number " + str(number) + "\n"

        #write the line to the file
        file.write(line)

    #close the file
    file.close()
```

B.

```
#display the entire contents of a file
def readWholeFile():
        #open a file called 'temp.txt' in read mode
        file = open("temp.txt","r")

        #read and print the entire contents of the file in one go
        print(file.read())

        #close the file
        file.close()

    #display the contents of a file line by line
    def readFileLineByLine():
        #open a file called 'temp.txt' in read mode
        file = open("temp.txt","r")

        #read and print one line at a time until file finished
        line = file.readline()
```

```
while line != "":
    print(line)
    line = file.readline()

#close file when finished
file.close()
```

Assessment/Activities:

1. Which of the following command is used to open a file "c:\temp.txt" in read-mode only?

a. infile = open("c:\temp.txt", "r")
b. infile = open("c:\\temp.txt", "r")
c. infile = open(file = "c:\temp.txt", "r+")
d. infile = open(file = "c:\\temp.txt", "r+")

2. Which of the following command is used to open a file "c:\temp.txt" in write-mode only?

a. outfile = open("c:\temp.txt", "w")
b. outfile = open("c:\\temp.txt", "w")
c. outfile = open(file = "c:\temp.txt", "w+")
d. outfile = open(file = "c:\\temp.txt", "w+")

3. Which of the following command is used to open a file "c:\temp.txt" in append-mode?

a. outfile = open("c:/temp.txt", "a")
b. outfile = open("c:\\temp.txt", "rw")
c. outfile = open("c:\temp.txt", "w+")
d. outfile = open("c:\\temp.txt", "r+")
e. outfile = open("c:\\temp.txt", "a")

4. Which of the following statements are true regarding the opening modes of a file?

a. when you open a file for reading, if the file does not exist, an error occurs.
b. when you open a file for writing, if the file does not exist, an error occurs.
c. when you open a file for reading, if the file does not exist, the program will open an empty file.
d. when you open a file for writing, if the file does not exist, a new file is created.
e. when you open a file for writing, if the file exists, the existing file is overwritten with the new file.

5. Which of the following commands can be used to read “n” number of characters from a file using the file object <file>?

a. file.read(n)
b. n = file.read()
c. file.readline(n)
d. file.readlines()

6. Which of the following commands can be used to read the entire contents of a file as a string using the file object <tmpfile>?

a. tmpfile.read(n)
b. tmpfile.read()
c. tmpfile.readline()
d. tmpfile.readlines()

7. Which of the following commands can be used to read the next line in a file using the file object <tmpfile>?

a. tmpfile.read(n)
b. tmpfile.read()
c. tmpfile.readline()
d. tmpfile.readlines()

8. Which of the following commands can be used to read the remaining lines in a file using the file object <tmpfile>?

a. tmpfile.read(n)
b. tmpfile.read()
c. tmpfile.readline()
d. tmpfile.readlines()

9. What does the <readlines()> method returns?

a. str
b. a list of lines
c. list of single characters
d. list of integers

10. Which of the following functions displays a file dialog for opening an existing file?

a. tmpfile = askopenfilename()
b. tmpfile = asksaveasfilename()
c. tmpfile = openfilename()
d. tmpfile = saveasfilename()

Lesson 7 – Exception Handling

Objectives

At the end of lesson, the students will have a knowledge in:

- Apply Exception Handling using Python
- Demonstrate an understanding of Exception Handling to different programming tasks
- Develop manipulation skills using exception handling

General Course Materials

- Python Tutorials for Beginners https://www.guru99.com/python-tutorials.html
- Python 3 https://www.tutorialspoint.com/python3/python_tutorial.pdf
- Python Full Course - Learn Python in 12 Hours | Python Tutorial For Beginners | Edureka https://www.youtube.com/watch?v=WGJJIrtnfpk

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them −

- **Exception Handling** − This would be covered in this tutorial. Here is a list standard Exceptions available in Python: Standard Exceptions.
- **Assertions** − This would be covered in Assertions in Python tutorial.

List of Standard Exceptions

Exception Name & Description
Exception Base class for all exceptions
StopIteration Raised when the next() method of an iterator does not point to any object.
SystemExit Raised by the sys.exit() function.
StandardError Base class for all built-in exceptions except StopIteration and SystemExit.
ArithmeticError Base class for all errors that occur for numeric calculation.
OverflowError Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError Raised when a floating point calculation fails.
ZeroDivisionError Raised when division or modulo by zero takes place for all numeric types.

AssertionError Raised in case of failure of the Assert statement.
AttributeError Raised in case of failure of attribute reference or assignment.
EOFError Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError Raised when an import statement fails.
KeyboardInterrupt Raised when the user interrupts program execution, usually by pressing Ctrl+c.
LookupError Base class for all lookup errors.
IndexError Raised when an index is not found in a sequence.
KeyError Raised when the specified key is not found in the dictionary.
NameError Raised when an identifier is not found in the local or global namespace.
UnboundLocalError Raised when trying to access a local variable in a function or method but no value has been assigned to it.
EnvironmentError Base class for all exceptions that occur outside the Python environment.
IOError Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
IOError Raised for operating system-related errors.
SyntaxError Raised when there is an error in Python syntax.
IndentationError Raised when indentation is not specified properly.
SystemError Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
SystemExit Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.

TypeError Raised when an operation or function is attempted that is invalid for the specified data type.
ValueError Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError Raised when a generated error does not fall into any category.
NotImplementedError Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

Assertions in Python

An assertion is a sanity-check that you can turn on or turn off when you are done with your testing of the program.

The easiest way to think of an assertion is to liken it to a **raise-if** statement (or to be more accurate, a raise-if-not statement). An expression is tested, and if the result comes up false, an exception is raised.

Assertions are carried out by the assert statement, the newest keyword to Python, introduced in version 1.5.

Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output.

> **The *assert* Statement**
> When it encounters an assert statement, Python evaluates the accompanying expression, which is hopefully true. If the expression is false, Python raises an *AssertionError* exception.

What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error. When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an exception

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

- **The *except* Clause with No Exceptions**

You can also use the except statement with no exceptions
This kind of a **try-except** statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

- **The *except* Clause with Multiple Exceptions**

You can also use the same *except* statement to handle multiple exceptions.

- **The try-finally Clause**

You can use a **finally:** block along with a **try:** block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not.

Argument of an Exception

An exception can have an *argument*, which is a value that gives additional information about the problem. The contents of the argument vary by exception. You capture an exception's argument by supplying a variable in the except clause
If you write the code to handle a single exception, you can have a variable follow the name of the exception in the except statement. If you are trapping multiple exceptions, you can have a variable follow the tuple of the exception.
This variable receives the value of the exception mostly containing the cause of the exception. The variable can receive a single value or multiple values in the form of a tuple. This tuple usually contains the error string, the error number, and an error location.

User-Defined Exceptions

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.
Here is an example related to *RuntimeError*. Here, a class is created that is subclassed from *RuntimeError*. This is useful when you need to display more specific information when an exception is caught.
In the try block, the user-defined exception is raised and caught in the except block. The variable e is used to create an instance of the class *Networkerror*.

Activities/Assessments:

1. An exception is the type of error that is detected:

- during syntax check
- during compile
- at runtime

during static analysis

2. One reason for data validation is to check that:

   - code meets the requirements
   - data has valid values
   - code has no syntax errors
   - data has security checks

3. Which best describes the code fragment:

```
result = action(num)
if result < 0

    raise InvalidAction('There is no such action')
```

   - It raises an exception
   - It catches an exception
   - It creates default exception
   - It loops until there is an exception
   -

4. How many except statements can a try-except block have?

   - One
   - Zero
   - More than zero
   - More than one

5. When is the finally block statements executed?

   - When a specific condition is specified
   - When there is an exception in try block
   - When there is no exception in try block
   - always

Lesson 8 – Introduction to Object-Oriented

Objectives

At the end of lesson, the students will have a knowledge in:

- Learn the concept of Object-oriented approach using Python
- Demonstrate an understanding of Object-Oriented Approach
- Develop Object-oriented approach in Python

General Course Materials

- Python Tutorials for Beginners https://www.guru99.com/python-tutorials.html
- Python 3 https://www.tutorialspoint.com/python3/python_tutorial.pdf
- Python Full Course - Learn Python in 12 Hours | Python Tutorial For Beginners | Edureka https://www.youtube.com/watch?v=WGJJIrtnfpk

Overview of OOP Terminology

- **Class** − A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable** − A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- **Data member** − A class variable or instance variable that holds data associated with a class and its objects.
- **Function overloading** − The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.
- **Instance variable** − A variable that is defined inside a method and belongs only to the current instance of a class.
- **Inheritance** − The transfer of the characteristics of a class to other classes that are derived from it.
- **Instance** − An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- **Instantiation** − The creation of an instance of a class.
- **Method** − A special kind of function that is defined in a class definition.
- **Object** − A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- **Operator overloading** − The assignment of more than one function to a particular operator.

Creating Classes

The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon

- The class has a documentation string, which can be accessed via *ClassName.__doc__*.
- The *class_suite* consists of all the component statements defining class members, data attributes and functions.

Creating Instance Objects

To create instances of a class, you call the class using class name and pass in whatever arguments its __*init*__ method accepts.

Accessing Attributes

You access the object's attributes using the dot operator with object. Class variable would be accessed using class name
Instead of using the normal statements to access attributes, you can use the following functions −

- The **getattr(obj, name[, default])** − to access the attribute of object.
- The **hasattr(obj,name)** − to check if an attribute exists or not.
- The **setattr(obj,name,value)** − to set an attribute. If attribute does not exist, then it would be created.
- The **delattr(obj, name)** − to delete an attribute.

Built-In Class Attributes

Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute −

- __**dict**__ − Dictionary containing the class's namespace.
- __**doc**__ − Class documentation string or none, if undefined.
- __**name**__ − Class name.
- __**module**__ − Module name in which the class is defined. This attribute is "__main__" in interactive mode.
- __**bases**__ − A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

Destroying Objects (Garbage Collection)

Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space. The process by which Python periodically reclaims blocks of memory that no longer are in use is termed Garbage Collection.

Python's garbage collector runs during program execution and is triggered when an object's reference count reaches zero. An object's reference count changes as the number of aliases that point to it changes.

An object's reference count increases when it is assigned a new name or placed in a container (list, tuple, or dictionary). The object's reference count decreases when it's deleted with *del*, its reference is reassigned, or its reference goes out of scope. When an object's reference count reaches zero, Python collects it automatically.

Class Inheritance

Instead of starting from scratch, you can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.

The child class inherits the attributes of its parent class, and you can use those attributes as if they were defined in the child class. A child class can also override data members and methods from the parent.

Overriding Methods

You can always override your parent class methods. One reason for overriding parent's methods is because you may want special or different functionality in your subclass.

Base Overloading Methods

Following table lists some generic functionality that you can override in your own classes −

Method, Description & Sample Call
__init__ (self [,args...]) Constructor (with any optional arguments) Sample Call : *obj = className(args)*
__del__(self) Destructor, deletes an object Sample Call : *del obj*
__repr__(self) Evaluable string representation Sample Call : *repr(obj)*
__str__(self) Printable string representation Sample Call : *str(obj)*
__cmp__ (self, x) Object comparison Sample Call : *cmp(obj, x)*

Overloading Operators

Suppose you have created a Vector class to represent two-dimensional vectors, what happens when you use the plus operator to add them? Most likely Python will yell at you.

You could, however, define the *__add__* method in your class to perform vector addition and then the plus operator would behave as per expectation

Data Hiding

An object's attributes may or may not be visible outside the class definition. You need to name attributes with a double underscore prefix, and those attributes then are not be directly visible to outsiders.

Activities/Assessments:

1.What will be the output of the following Python code?

```
class test:
    def __init__(self,a="Hello World"):
        self.a=a

    def display(self):
        print(self.a)
obj=test()
obj.display()
```

a) The program has an error because constructor can't have default arguments
b) Nothing is displayed
c) "Hello World" is displayed
d) The program has an error display function doesn't have parameters

2. What is setattr() used for?
a) To access the attribute of the object
b) To set an attribute
c) To check if an attribute exists or not
d) To delete an attribute

3.What will be the output of the following Python code?

```
class change:
    def __init__(self, x, y, z):
        self.a = x + y + z

x = change(1,2,3)
y = getattr(x, 'a')
setattr(x, 'a', y+1)
print(x.a)
```

a) 6
b) 7
c) Error
d) 0

4. What will be the output of the following Python code?

```
class test:
    def __init__(self,a):
        self.a=a
```

```
    def display(self):
        print(self.a)
obj=test()
obj.display()
```

a) Runs normally, doesn't display anything
b) Displays 0, which is the automatic default value
c) Error as one argument is required while creating the object
d) Error as display function requires additional argument

5. Is the following Python code correct?

```
>>> class A:
    def __init__(self,b):
            self.b=b
    def display(self):
            print(self.b)
>>> obj=A("Hello")
>>> del obj
```

a) True
b) False

Lesson 9 – GUI Programming (Tkinter)

Objectives

At the end of lesson, the students will have a knowledge in:

- Learn the Graphical User Interface (GUI) Programming
- Demonstrate an understanding of Graphical User Interface (GUI) Programming
- Develop manipulation skills for Graphical User Interface (GUI) Programming in Python

General Course Materials

- Python Tutorials for Beginners https://www.guru99.com/python-tutorials.html
- Python 3 https://www.tutorialspoint.com/python3/python_tutorial.pdf
- Python Full Course - Learn Python in 12 Hours | Python Tutorial For Beginners | Edureka https://www.youtube.com/watch?v=WGJJIrtnfpk

Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below.

- **Tkinter** − Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look this option in this chapter.
- **wxPython** − This is an open-source Python interface for wxWindows http://wxpython.org.
- **JPython** − JPython is a Python port for Java which gives Python scripts seamless access to Java class libraries on the local machine http://www.jython.org.

There are many other interfaces available, which you can find them on the net.

Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps −

- Import the *Tkinter* module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

Operator & Description
Button The Button widget is used to display buttons in your application.

Canvas The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
Checkbutton The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
Entry The Entry widget is used to display a single-line text field for accepting values from a user.
Frame The Frame widget is used as a container widget to organize other widgets.
Label The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
Listbox The Listbox widget is used to provide a list of options to a user.
Menubutton The Menubutton widget is used to display menus in your application.
Menu The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
Message The Message widget is used to display multiline text fields for accepting values from a user.
Radiobutton The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
Scale The Scale widget is used to provide a slider widget.
Scrollbar The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
Text The Text widget is used to display text in multiple lines.
Toplevel The Toplevel widget is used to provide a separate window container.
Spinbox The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.
PanedWindow A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.

LabelFrame A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
tkMessageBox This module is used to display message boxes in your applications.

Standard attributes

Let us take a look at how some of their common attributes.such as sizes, colors and fonts are specified.

- Dimensions
- Colors
- Fonts
- Anchors
- Relief styles
- Bitmaps
- Cursors

Geometry Management

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

- The *pack()* Method − This geometry manager organizes widgets in blocks before placing them in the parent widget.
- The *grid()* Method − This geometry manager organizes widgets in a table-like structure in the parent widget.
- The *place()* Method − This geometry manager organizes widgets by placing them in a specific position in the parent widget.

Assessment/Activities:

1. Config() in Python Tkinter are used for
    - 1.destroy the widget
    - 2.place the widget
    - 3.change property of the widget
    - 4.configure the widge
2. Correct way to draw a line in canvas tkinter ?
    - 1.line()
    - 2.canvas.create_line()
    - 3.create_line(canvas)
    - 4.None of the above
3. Creating line are come in which type of thing ?

- GUI
- Canvas
- Both of the above
- None of the above

4. Essential thing to create a window screen using tkinter python?

   - call tk() function
   - create a button
   - To define a geometry
   - All of the above

5. fg in tkinter widget is stands for ?

   - 1.foreground
   - 2.background
   - 3.forgap
   - 4.None of the abov

6. How pack() function works on tkinter widget ?

   - 1.According to x,y coordinate
   - 2.According to row and column vise
   - 3.According to left,right,up,down
   - 4.None of the above

7. In which of the following field, we can put our Button?

   - 1.Window
   - 2.Frame
   - 3.Label
   - 4.All of the above

Lesson 10 – Database Programming (MySQL)

Objectives

At the end of lesson, the students will have a knowledge in:

- Learn and Understand the Database Programming using MYSQL
- Demonstrate the Database programming with MYSQL
- Develop manipulation skills in Database Programming in Python

General Course Materials

4. Python Tutorials for Beginners https://www.guru99.com/python-tutorials.html
5. Python 3 https://www.tutorialspoint.com/python3/python_tutorial.pdf
6. Python Full Course - Learn Python in 12 Hours | Python Tutorial For Beginners | Edureka https://www.youtube.com/watch?v=WGJJIrtnfpk

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.
You can choose the right database for your application. Python Database API supports a wide range of database servers such as −

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase

Here is the list of available Python database interfaces: Python Database Interfaces and APIs. You must download a separate DB API module for each database you need to access. For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database modules.

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following −

- Importing the API module.
- Acquiring a connection with the database.
- Issuing SQL statements and stored procedures.
- Closing the connection

We would learn all the concepts using MySQL, so let us talk about MySQLdb module.

What is MySQLdb?

MySQLdb is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and is built on top of the MySQL C API.

How to Install MySQLdb?

Before proceeding, you make sure you have MySQLdb installed on your machine. Just type the following in your Python script and execute it

Database Connection

Before connecting to a MySQL database, make sure of the followings −

- You have created a database TESTDB.
- You have created a table EMPLOYEE in TESTDB.
- This table has fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.
- User ID "testuser" and password "test123" are set to access TESTDB.
- Python module MySQLdb is installed properly on your machine.
- You have gone through MySQL tutorial to understand MySQL Basics.

Creating Database Table

Once a database connection is established, we are ready to create tables or records into the database tables using **execute** method of the created cursor.

INSERT Operation

It is required when you want to create your records into a database table.

READ Operation

READ Operation on any database means to fetch some useful information from the database.
Once our database connection is established, you are ready to make a query into this database. You can use either **fetchone()** method to fetch single record or **fetchall()** method to fetech multiple values from a database table.

- **fetchone()** − It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **fetchall()** − It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
- **rowcount** − This is a read-only attribute and returns the number of rows that were affected by an execute() method.

UPDATE Operation

UPDATE Operation on any database means to update one or more records, which are already available in the database.

The following procedure updates all the records having SEX as **'M'**. Here, we increase AGE of all the males by one year.

DELETE Operation

DELETE operation is required when you want to delete some records from your database.
Performing Transactions

Transactions are a mechanism that ensures data consistency. Transactions have the following four properties −

- **Atomicity** − Either a transaction completes or nothing happens at all.
- **Consistency** − A transaction must start in a consistent state and leave the system in a consistent state.
- **Isolation** − Intermediate results of a transaction are not visible outside the current transaction.
- **Durability** − Once a transaction was committed, the effects are persistent, even after a system failure.

The Python DB API 2.0 provides two methods to either *commit* or *rollback* a transaction.

COMMIT Operation

Commit is the operation, which gives a green signal to database to finalize the changes, and after this operation, no change can be reverted back.

ROLLBACK Operation

If you are not satisfied with one or more of the changes and you want to revert back those changes completely, then use **rollback()** method.

Disconnecting Database

To disconnect Database connection, use close() method.

Handling Errors

There are many sources of errors. A few examples are a syntax error in an executed SQL statement, a connection failure, or calling the fetch method for an already cancelled or finished statement handle.

The DB API defines a number of errors that must exist in each database module. The following table lists these exceptions.

Exception & Description
Warning Used for non-fatal issues. Must subclass StandardError.
Error Base class for errors. Must subclass StandardError.
InterfaceError Used for errors in the database module, not the database itself. Must subclass Error.
DatabaseError Used for errors in the database. Must subclass Error.
DataError Subclass of DatabaseError that refers to errors in the data.

OperationalError Subclass of DatabaseError that refers to errors such as the loss of a connection to the database. These errors are generally outside of the control of the Python scripter.
IntegrityError Subclass of DatabaseError for situations that would damage the relational integrity, such as uniqueness constraints or foreign keys.
InternalError Subclass of DatabaseError that refers to errors internal to the database module, such as a cursor no longer being active.
ProgrammingError Subclass of DatabaseError that refers to errors such as a bad table name and other things that can safely be blamed on you.
NotSupportedError Subclass of DatabaseError that refers to trying to call unsupported functionality.

Your Python scripts should handle these errors, but before using any of the above exceptions, make sure your MySQLdb has support for that exception. You can get more information about them by reading the DB API 2.0 specification.

Connecting PyCharm to MS Access Database

1. **Determine the bit version of your python**

```
1  import struct
2  print(struct.calcsize("P") * 8)
```

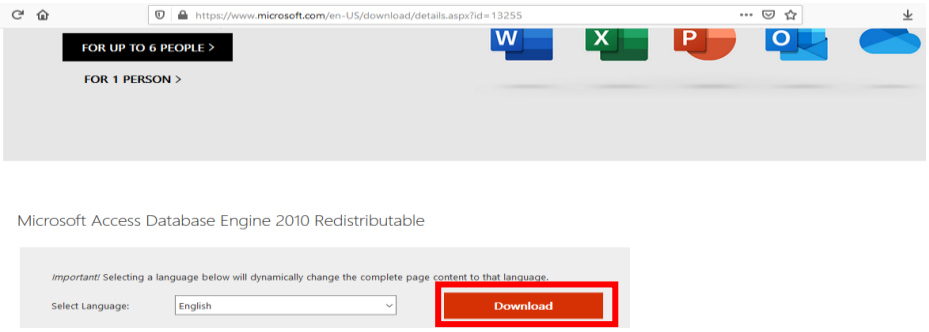
**Execute this code to know if your python is 32-bit or 64-bit*

OUTPUT:

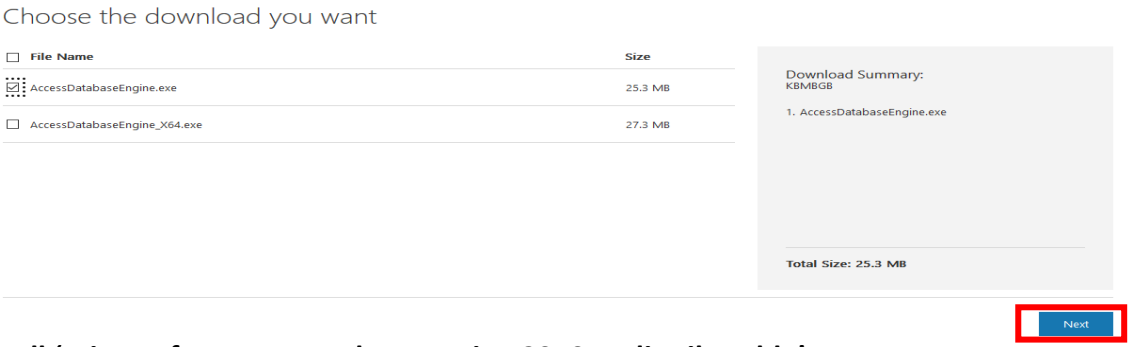
```
32

Process finished with exit code 0
```

**It means I'm using a 32-bit version of python*

2. **Download 'Microsoft Access Database Engine 2010 Redistributable'**
   a. https://www.microsoft.com/en-US/download/details.aspx?id=13255

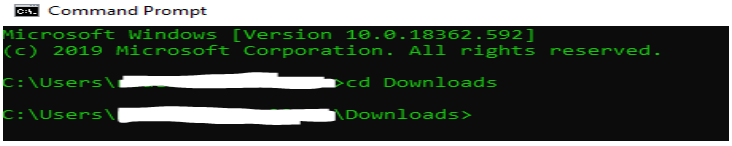


b. Choose 'AccessDatabaseEngine.exe' if your python is 32-bit, and 'AccessDatabaseEngine_X64.exe for 64-bit then click 'Next'

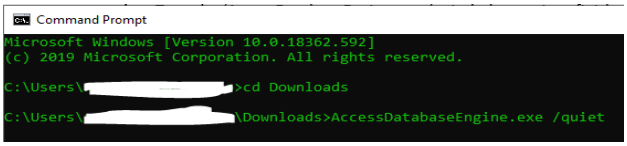


3. Install 'Microsoft Access Database Engine 2010 Redistributable'

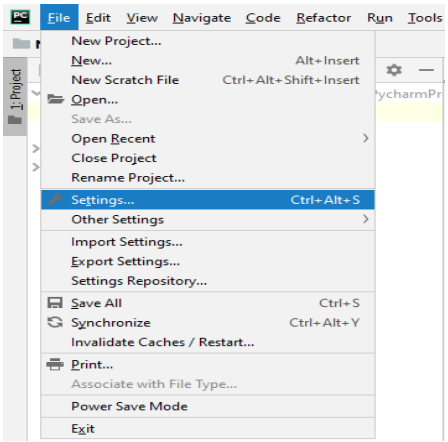
a. Open 'Command Prompt' and enter the path file of the AccessDatabaseEngine.exe



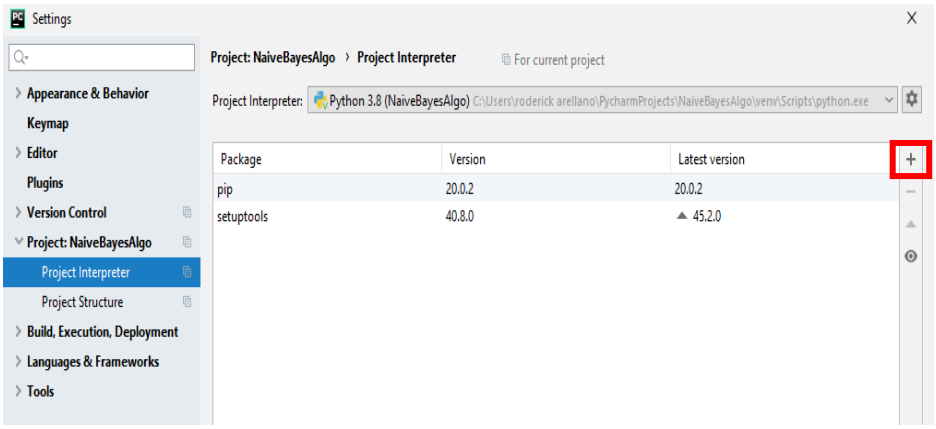
b. Type the 'AccessDatabaseEngine.exe /quiet', then wait to finish the installation



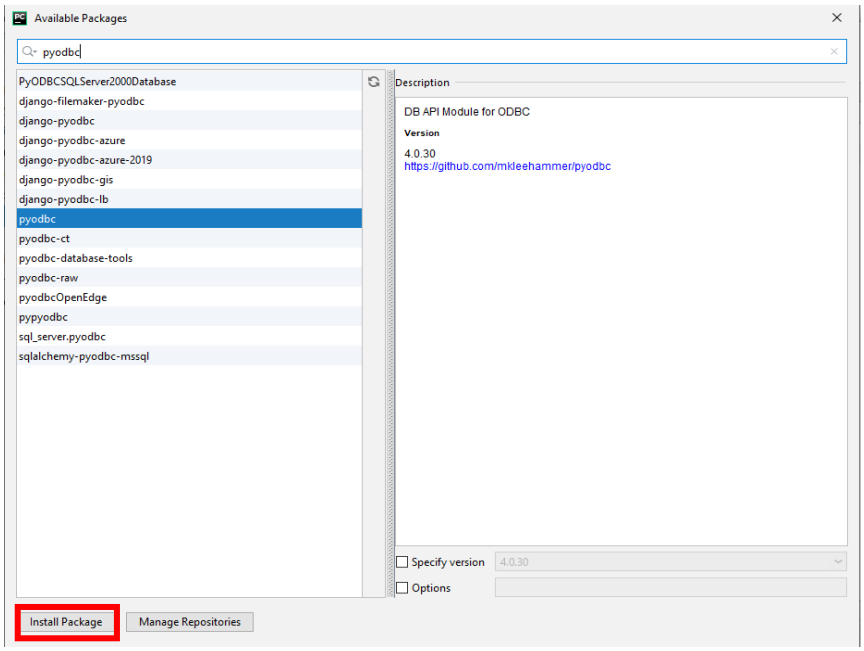
4. Install "pyodbc" in your project



***Click 'File', then click 'Settings'*

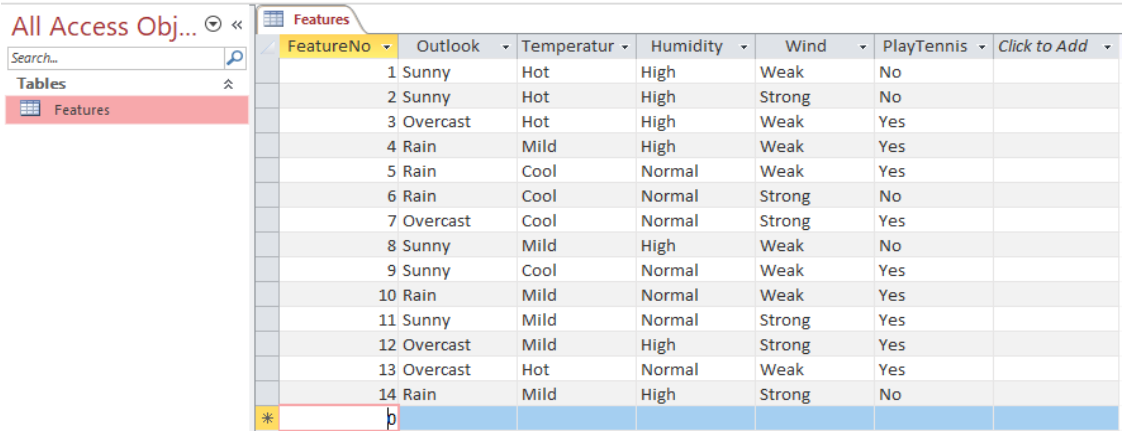


***Click the '+'*



**Search for 'pyodbc', then click 'Install Package'*

5. Create a database in MSaccess called 'PlayTennis.accdb' with a table named 'Features'



FeatureNo	Outlook	Temperatur	Humidity	Wind	PlayTennis	Click to Add
1	Sunny	Hot	High	Weak	No	
2	Sunny	Hot	High	Strong	No	
3	Overcast	Hot	High	Weak	Yes	
4	Rain	Mild	High	Weak	Yes	
5	Rain	Cool	Normal	Weak	Yes	
6	Rain	Cool	Normal	Strong	No	
7	Overcast	Cool	Normal	Strong	Yes	
8	Sunny	Mild	High	Weak	No	
9	Sunny	Cool	Normal	Weak	Yes	
10	Rain	Mild	Normal	Weak	Yes	
11	Sunny	Mild	Normal	Strong	Yes	
12	Overcast	Mild	High	Strong	Yes	
13	Overcast	Hot	Normal	Weak	Yes	
14	Rain	Mild	High	Strong	No	

6. Syntax for retrieving the records in MS Access database

Code

```
import pyodbc
conn_str = (
    r'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};'
    r'DBQ=C:\Users\\Student\Documents\PlayTennis.accdb;'
    )
conn = pyodbc.connect(conn_str)
cursor = conn.cursor()
query = "SELECT * FROM Features"
cursor.execute(query)
results = cursor.fetchall()
for result in results:
    print(result)
```

Output

```
(1, 'Sunny', 'Hot', 'High', 'Weak', 'No')
(2, 'Sunny', 'Hot', 'High', 'Strong', 'No')
(3, 'Overcast', 'Hot', 'High', 'Weak', 'Yes')
(4, 'Rain', 'Mild', 'High', 'Weak', 'Yes')
(5, 'Rain', 'Cool', 'Normal', 'Weak', 'Yes')
(6, 'Rain', 'Cool', 'Normal', 'Strong', 'No')
(7, 'Overcast', 'Cool', 'Normal', 'Strong', 'Yes')
(8, 'Sunny', 'Mild', 'High', 'Weak', 'No')
(9, 'Sunny', 'Cool', 'Normal', 'Weak', 'Yes')
(10, 'Rain', 'Mild', 'Normal', 'Weak', 'Yes')
(11, 'Sunny', 'Mild', 'Normal', 'Strong', 'Yes')
(12, 'Overcast', 'Mild', 'High', 'Strong', 'Yes')
(13, 'Overcast', 'Hot', 'Normal', 'Weak', 'Yes')
(14, 'Rain', 'Mild', 'High', 'Strong', 'No')

Process finished with exit code 0
```

7. Syntax for UPDATING a record in MS Access database

CODE

```
1  import pyodbc
2  conn_str = (
3      r'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};'
4      r'DBQ=C:\Users\\Student\Documents\PlayTennis.accdb;'
5      )
6  conn = pyodbc.connect(conn_str)
7  cursor = conn.cursor()
8
9  query = "SELECT * FROM Features"
10 cursor.execute(query)
11 results = cursor.fetchall()
12 print('\nRECORDS BEFORE UPDATING:')
13 for result in results:
14     print(result)
15
16 query = "UPDATE Features SET PlayTennis = 'Yes' WHERE FeatureNo = 1"
17 cursor.execute(query)
18 conn.commit()
19 print('\nUPDATED SUCCESSFULLY!')
20
21 query = "SELECT * FROM Features"
22 cursor.execute(query)
23 results = cursor.fetchall()
24 print('\nRECORDS AFTER UPDATING:')
25 for result in results:
26     print(result)
```

**Always check the file path of your MS access database (line 4)*

OUTPUT

```
RECORDS BEFORE UPDATING:
(1, 'Sunny', 'Hot', 'High', 'Weak', 'No')
(2, 'Sunny', 'Hot', 'High', 'Strong', 'No')
(3, 'Overcast', 'Hot', 'High', 'Weak', 'Yes')
(4, 'Rain', 'Mild', 'High', 'Weak', 'Yes')
(5, 'Rain', 'Cool', 'Normal', 'Weak', 'Yes')
(6, 'Rain', 'Cool', 'Normal', 'Strong', 'No')
(7, 'Overcast', 'Cool', 'Normal', 'Strong', 'Yes')
(8, 'Sunny', 'Mild', 'High', 'Weak', 'No')
(9, 'Sunny', 'Cool', 'Normal', 'Weak', 'Yes')
(10, 'Rain', 'Mild', 'Normal', 'Weak', 'Yes')
(11, 'Sunny', 'Mild', 'Normal', 'Strong', 'Yes')
(12, 'Overcast', 'Mild', 'High', 'Strong', 'Yes')
(13, 'Overcast', 'Hot', 'Normal', 'Weak', 'Yes')
(14, 'Rain', 'Mild', 'High', 'Strong', 'No')

UPDATED SUCCESSFULLY!
```

```
RECORDS AFTER UPDATING:
(1, 'Sunny', 'Hot', 'High', 'Weak', 'Yes')
(2, 'Sunny', 'Hot', 'High', 'Strong', 'No')
(3, 'Overcast', 'Hot', 'High', 'Weak', 'Yes')
(4, 'Rain', 'Mild', 'High', 'Weak', 'Yes')
(5, 'Rain', 'Cool', 'Normal', 'Weak', 'Yes')
(6, 'Rain', 'Cool', 'Normal', 'Strong', 'No')
(7, 'Overcast', 'Cool', 'Normal', 'Strong', 'Yes')
(8, 'Sunny', 'Mild', 'High', 'Weak', 'No')
(9, 'Sunny', 'Cool', 'Normal', 'Weak', 'Yes')
(10, 'Rain', 'Mild', 'Normal', 'Weak', 'Yes')
(11, 'Sunny', 'Mild', 'Normal', 'Strong', 'Yes')
(12, 'Overcast', 'Mild', 'High', 'Strong', 'Yes')
(13, 'Overcast', 'Hot', 'Normal', 'Weak', 'Yes')
(14, 'Rain', 'Mild', 'High', 'Strong', 'No')

Process finished with exit code 0
```

8. Syntax for INSERTING a record in MS Access database

CODE

```
1  import pyodbc
2  conn_str = (
3      r'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};'
4      r'DBQ=C:\Users\\Student\Documents\PlayTennis.accdb;'
5      )
6  conn = pyodbc.connect(conn_str)
7  cursor = conn.cursor()
8
9  query = "SELECT * FROM Features"
10 cursor.execute(query)
11 results = cursor.fetchall()
12 print('\nRECORDS BEFORE INSERTING:')
13 for result in results:
14     print(result)
15
16 query = "INSERT into Features (FeatureNo, Outlook, Temperature, Humidity, Wind, PlayTennis)" \
17         " VALUES (15, 'Sunny', 'Hot', 'High', 'Weak', 'Yes')"
18 cursor.execute(query)
19 conn.commit()
20 print('\nINSERTED SUCCESSFULLY!')
21
22 query = "SELECT * FROM Features"
23 cursor.execute(query)
24 results = cursor.fetchall()
25 print('\nRECORDS AFTER INSERTING:')
26 for result in results:
27     print(result)
```

**Always check the file path of your MS access database (line 4)*

OUTPUT

```
RECORDS BEFORE INSERTING:
(1, 'Sunny', 'Hot', 'High', 'Weak', 'No')
(2, 'Sunny', 'Hot', 'High', 'Strong', 'No')
(3, 'Overcast', 'Hot', 'High', 'Weak', 'Yes')
(4, 'Rain', 'Mild', 'High', 'Weak', 'Yes')
(5, 'Rain', 'Cool', 'Normal', 'Weak', 'Yes')
(6, 'Rain', 'Cool', 'Normal', 'Strong', 'No')
(7, 'Overcast', 'Cool', 'Normal', 'Strong', 'Yes')
(8, 'Sunny', 'Mild', 'High', 'Weak', 'No')
(9, 'Sunny', 'Cool', 'Normal', 'Weak', 'Yes')
(10, 'Rain', 'Mild', 'Normal', 'Weak', 'Yes')
(11, 'Sunny', 'Mild', 'Normal', 'Strong', 'Yes')
(12, 'Overcast', 'Mild', 'High', 'Strong', 'Yes')
(13, 'Overcast', 'Hot', 'Normal', 'Weak', 'Yes')
(14, 'Rain', 'Mild', 'High', 'Strong', 'No')

INSERTED SUCCESSFULLY!

RECORDS AFTER INSERTING:
(1, 'Sunny', 'Hot', 'High', 'Weak', 'No')
(2, 'Sunny', 'Hot', 'High', 'Strong', 'No')
(3, 'Overcast', 'Hot', 'High', 'Weak', 'Yes')
(4, 'Rain', 'Mild', 'High', 'Weak', 'Yes')
(5, 'Rain', 'Cool', 'Normal', 'Weak', 'Yes')
(6, 'Rain', 'Cool', 'Normal', 'Strong', 'No')
(7, 'Overcast', 'Cool', 'Normal', 'Strong', 'Yes')
(8, 'Sunny', 'Mild', 'High', 'Weak', 'No')
(9, 'Sunny', 'Cool', 'Normal', 'Weak', 'Yes')
(10, 'Rain', 'Mild', 'Normal', 'Weak', 'Yes')
(11, 'Sunny', 'Mild', 'Normal', 'Strong', 'Yes')
(12, 'Overcast', 'Mild', 'High', 'Strong', 'Yes')
(13, 'Overcast', 'Hot', 'Normal', 'Weak', 'Yes')
(14, 'Rain', 'Mild', 'High', 'Strong', 'No')
(15, 'Sunny', 'Hot', 'High', 'Weak', 'Yes')

Process finished with exit code 0
```

9. Syntax for DELETING a record in MS Access database

```
1   import pyodbc
2   conn_str = (
3       r'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};'
4       r'DBQ=C:\Users\\Student\Documents\PlayTennis.accdb;'
5       )
6   conn = pyodbc.connect(conn_str)
7   cursor = conn.cursor()
8
9   query = "SELECT * FROM Features"
10  cursor.execute(query)
11  results = cursor.fetchall()
12  print('\nRECORDS BEFORE DELETING:')
13  for result in results:
14      print(result)
15
16  query = "DELETE FROM Features WHERE FeatureNo = 15"
17  cursor.execute(query)
18  conn.commit()
19  print('\nDELETED SUCCESSFULLY!')
20
21  query = "SELECT * FROM Features"
22  cursor.execute(query)
23  results = cursor.fetchall()
24  print('\nRECORDS AFTER DELETING:')
25  for result in results:
26      print(result)
```

**Always check the file path of your MS access database (line 4)*

OUTPUT

```
RECORDS BEFORE DELETING:
(1, 'Sunny', 'Hot', 'High', 'Weak', 'No')
(2, 'Sunny', 'Hot', 'High', 'Strong', 'No')
(3, 'Overcast', 'Hot', 'High', 'Weak', 'Yes')
(4, 'Rain', 'Mild', 'High', 'Weak', 'Yes')
(5, 'Rain', 'Cool', 'Normal', 'Weak', 'Yes')
(6, 'Rain', 'Cool', 'Normal', 'Strong', 'No')
(7, 'Overcast', 'Cool', 'Normal', 'Strong', 'Yes')
(8, 'Sunny', 'Mild', 'High', 'Weak', 'No')
(9, 'Sunny', 'Cool', 'Normal', 'Weak', 'Yes')
(10, 'Rain', 'Mild', 'Normal', 'Weak', 'Yes')
(11, 'Sunny', 'Mild', 'Normal', 'Strong', 'Yes')
(12, 'Overcast', 'Mild', 'High', 'Strong', 'Yes')
(13, 'Overcast', 'Hot', 'Normal', 'Weak', 'Yes')
(14, 'Rain', 'Mild', 'High', 'Strong', 'No')
(15, 'Sunny', 'Hot', 'High', 'Weak', 'Yes')

DELETED SUCCESSFULLY!
```

```
RECORDS AFTER DELETING:
(1, 'Sunny', 'Hot', 'High', 'Weak', 'No')
(2, 'Sunny', 'Hot', 'High', 'Strong', 'No')
(3, 'Overcast', 'Hot', 'High', 'Weak', 'Yes')
(4, 'Rain', 'Mild', 'High', 'Weak', 'Yes')
(5, 'Rain', 'Cool', 'Normal', 'Weak', 'Yes')
(6, 'Rain', 'Cool', 'Normal', 'Strong', 'No')
(7, 'Overcast', 'Cool', 'Normal', 'Strong', 'Yes')
(8, 'Sunny', 'Mild', 'High', 'Weak', 'No')
(9, 'Sunny', 'Cool', 'Normal', 'Weak', 'Yes')
(10, 'Rain', 'Mild', 'Normal', 'Weak', 'Yes')
(11, 'Sunny', 'Mild', 'Normal', 'Strong', 'Yes')
(12, 'Overcast', 'Mild', 'High', 'Strong', 'Yes')
(13, 'Overcast', 'Hot', 'Normal', 'Weak', 'Yes')
(14, 'Rain', 'Mild', 'High', 'Strong', 'No')

Process finished with exit code 0
```

Activities/Assessment:

1. **What is the output of the python program below?**

```
import mysql.connector
from mysql.connector import Error

try:
    connection = mysql.connector.connect(host='localhost',
                                         database='Electronics',
                                         user='pynative',
                                         password='pynative@#29')
    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Connected to MySQL Server version ", db_Info)
        cursor = connection.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()
        print("You're connected to database: ", record)

except Error as e:
    print("Error while connecting to MySQL", e)
finally:
    if connection.is_connected():
        cursor.close()
        connection.close()
        print("MySQL connection is closed")
```

Output

Lesson 11
Application of Data Analytics Using Python

Objectives

At the end of lesson, the students will have a knowledge in:

- To understand the application of Machine Learning Algorithm in Data Analytics
- To know the steps of Machine Learning Models
- To understand the concepts of a Classification

General Course Materials

- Watch Machine Learning with Python
- https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=2ahUKEwie_a_o5L3pAhXTEnAKHWkhBQQQwqsBMAJ6BAgMEAw&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DRnFGwxJwx-0&usg=AOvVaw2EicuCdwXKr2FrSNzrhkfO
- https://www.kdnuggets.com/2019/04/naive-bayes-baseline-model-machine-learning-classification-performance.html

Data analytics

It is the science of analyzing raw data in order to make conclusions about that information. Data analytics techniques can reveal trends and metrics that would otherwise be lost in the mass of information. This information can then be used to optimize processes to increase the overall efficiency of a business or system.

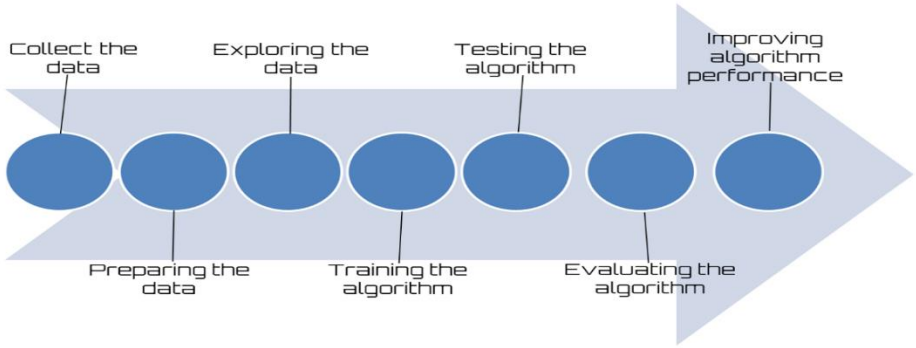
Data Analysis

Data analysis is defined as a process of cleaning, transforming, and modeling data to discover useful information for business decision-making. The purpose of Data Analysis is to extract useful information from data and taking the decision based upon the data analysis.

Machine Learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

Machine Learning Process



1 - Data Collection. The quantity & quality of your data dictate how accurate our model is. ...
2 - Data **Preparation**. Wrangle data and prepare it for training. ...
3 - Choose a Model. ...
4 - Train the Model. ...
5 - Evaluate the Model. ...
6 - Parameter Tuning. ...
7 - Make Predictions.
7 - Make Predictions

Naive Bayes Algorithm

- **Naive Bayes** is one of the simplest machine learning algorithms. It is supervised algorithm.
- Naive Bayes is a classification algorithm and is extremely fast. It uses Bayes theory of probability.

Why Naive?

- It is called 'naive' because the algorithm assumes that all attributes are independent of each other.
- Naive Bayes algorithm is commonly used in text classification with multiple classes.
- To understand how Naive Bayes algorithm works, it is important to understand Bayes theory of probability. Let's work through an example to derive Bayes theory.

Naive Bayes

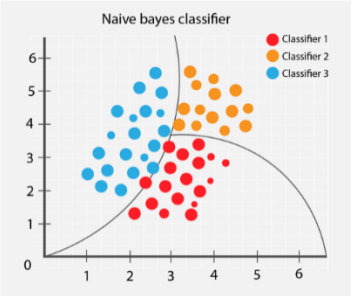
thatware.co

In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$



- P(A|B) - **Posterior Probability**
  - The conditional probability that event A occurs given that event B has occurred.
- P(A) - **Prior Probability**
  - The probability of event A.
- P(B) - **Evidence**
  - The probability of event B.
- P(B|A) - **Likelihood**
  - The conditional probability of B occurring given event A has occurred.

Python's Scikitlearn gives the user access to the following 3 Naive Bayes models.

- Gaussian
  - The gaussian NB Alogorithm assumes all contnuous features (predictors) and all follow a Gaussian (Normal Distribution).
- Multinomial
  - Multinomial NB is suited for discrete data that have frequencies and counts. Spam Filtering and Text/Document Classification are two very well-known use cases.
- Bernoulli
  - Bernoulli is similar to Multinomial except it is for boolean/binary features. Like the multinomial method it can be used for spam filtering and document classification in which binary terms (i.e. word occurrence in a document represented with True or False).

What is scikit-learn?

Scikit-learn is probably the most useful library for machine learning in Python. It is on NumPy, SciPy and matplotlib, this library contains a lot of effiecient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. Please note that scikit-learn is used to build models. It should not be used for reading the data, manipulating and summarizing it. There are better libraries for that (e.g. NumPy, Pandas etc.)

Components of scikit-learn:

Scikit-learn comes loaded with a lot of features. Here are a few of them to help you understand the spread:

Supervised learning algorithms:

Think of any supervised learning algorithm you might have heard about and there is a very high chance that it is part of scikit-learn. Starting from Generalized linear models (e.g Linear Regression), Support Vector Machines (SVM), Decision Trees to Bayesian methods – all of them are part of scikit-learn toolbox. The spread of algorithms is one of the big reasons for high usage of scikit-learn.

Cross-validation:

There are various methods to check the accuracy of supervised models on unseen data

Unsupervised learning algorithms:

Again there is a large spread of algorithms in the offering – starting from clustering, factor analysis, principal component analysis to unsupervised neural networks.

Various toy datasets: This came in handy while learning scikit-learn. I had learnt SAS using various academic datasets (e.g. IRIS dataset, Boston House prices dataset). Having them handy while learning a new library helped a lot.

Feature extraction: Useful for extracting features from images and text (e.g. Bag of words)

Community / Organizations using scikit-learn:

One of the main reasons behind using open source tools is the huge community it has. Same is true for scikit-learn as well. There are about 35 contributors to scikit learn till date, the most notable being Andreas Mueller (P.S. Andy's machine learning cheat sheet is one of the best visualizations to understand the spectrum of machine learning algorithms).

Quick Example:

Now that you understand the eco-system at a high level, let me illustrate the use of scikit learn with an example. The idea is to just illustrate the simplicity of usage of scikit-learn. We will have a look at various algorithms and best ways to use them in one of the articles which follow.
We will build a logistic regression on IRIS dataset:

Step 1: Import the relevant libraries and read the dataset

```
import numpy as np
import matplotlib as plt
from sklearn import datasets
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
```

We have imported all the libraries. Next, we read the dataset:

```
dataset = datasets.load_iris()
```

Step 2: Understand the dataset by looking at distributions and plots
I am skipping these steps for now. You can read this article, if you want to learn exploratory analysis.

Step 3: Build a logistic regression model on the dataset and making predictions

```
model.fit(dataset.data, dataset.target)
expected = dataset.target
predicted = model.predict(dataset.data)
```

Step 4: Print confusion metrix

```
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

Scikit-learn (formerly **scikits.learn** and also known as **sklearn**) is a free software machine learning library for the Python programming language.[3] It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Overview

The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy.[4] The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from the French Institute for Research in Computer Science and Automation in Rocquencourt, France, took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012. Scikit-learn is one of the most popular machine learning libraries on GitHub.

Implementation

Scikit-learn is largely written in Python, and uses numpy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible.

Scikit-learn integrates well with many other Python libraries, such as matplotlib and plotly for plotting, numpy for array vectorization, pandas dataframes, scipy, and many more.

Version history

Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007. Later Matthieu Brucher joined the project and started to use it as a part of his thesis work. In 2010 INRIA, the French Institute for Research in Computer Science and Automation, got involved and the first public release (v0.1 beta) was published in late January 2010.

- May 2019. scikit-learn 0.21.0[8]
- September 2018. scikit-learn 0.20.0[9]
- July 2017. scikit-learn 0.19.0
- September 2016. scikit-learn 0.18.0
- November 2015. scikit-learn 0.17.0[10]
- March 2015. scikit-learn 0.16.0[10]
- July 2014. scikit-learn 0.15.0[10]
- August 2013. scikit-learn 0.14[10]

Table 1: Sample Dataset to be modelled using Bayesian algorithm to predict whether to play golf or not.

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	normal	False	yes
10	sunny	mild	normal	True	yes
11	overcast	mild	high	True	yes
12	overcast	hot	normal	False	yes
13	rainy	mild	high	True	no

Sample Python programming with Bayesin algorithm using Scikit to build models for prediction

```
Importing all necessary libraries
import pandas as pd

#scikit-learn is used to build models. It should not be used for reading the data, manipulating and summarizing it.
#Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for
# the Python programming language.[3] It features various classification, regression and clustering algorithms
# including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to
# interoperate with the Python numerical and scientific libraries NumPy and SciPy.
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the CSV data file using pandas read_csv method
play_tennis = pd.read_csv("PlayTennis.csv")
print(play_tennis)

# LabelEncoder() converts a categorical data into a number ranging from 0 to n-1
# where n is the number of classes in the variable.
number = LabelEncoder()

# 3 classes of Outlook (Overcast, Rain, Sunny), these are represented as
# 0 - Overcast, 1 - Rain, 2 - Sunny
play_tennis['Outlook'] = number.fit_transform(play_tennis['Outlook'])

# 3 classes of Temperature (Cool, Hot, Mild), these are represented as
# 0 - Cool, 1 - Hot, 2 - Mild
play_tennis['Temperature'] = number.fit_transform(play_tennis['Temperature'])

# 2 classes of Humidity (High, Normal), these are represented as
# 0 - High, 1 - Normal
play_tennis['Humidity'] = number.fit_transform(play_tennis['Humidity'])

# 2 classes of Wind (Strong, Weak), these are represented as
# 0 - Strong, 1 - Weak
play_tennis['Wind'] = number.fit_transform(play_tennis['Wind'])

# 2 classes of Play Tennis (No, Yes), these are represented as
# 0 - No, 1 - Yes
play_tennis['Play Tennis'] = number.fit_transform(play_tennis['Play Tennis'])
print(play_tennis)
```

```
# Defining the features and the target variables
features = ["Outlook", "Temperature", "Humidity", "Wind"]
target = "Play Tennis"

# Create a train, test split. We build the model using the train dataset
# and we will validate the model on the test dataset
#if you use random_state=some_number, then you can guarantee that the output of Run 1 will be equal to the output of Run 2, i.e. your split will be always the same. It doesn't matter what the actual random_state number is 42, 0, 21,
#  The important thing is that everytime you use 42, you will always get the same output the first time you make the split.
#Data Slicing Let's split the data into training and test set. We can easily perform this step using sklearn's train_test_split() method.
features_train, features_test, target_train, target_test = train_test_split(play_tennis[features],
play_tennis[target], test_size = 0.30,
   random_state = 40)

# Displaying the split datasets
print('\tTraining Features\n ',features_train)
print('\tTesting Features\n ',features_test)
print('\tTraining Target\n ',target_train)
print('\tTesting Target\n ',target_test)
#Gaussian Naive Bayes Implementation .After completing the data preprocessing. it's time to implement machine learning
# algorithm on it. We are going to use sklearn's GaussianNB module.
# Creating a Gaussian Naive Bayes model
model = GaussianNB()
#We have built a GaussianNB classifier. The classifier is trained using training data.
# We can use fit() method for training it. After building a classifier, our model is ready to make predictions.
# We can use predict() method with test set features as its parameters.
# Fitting the training dataset to the model
model.fit(features_train, target_train)

# After fitting, we will make predictions using the testing dataset
pred = model.predict(features_test)
#It's time to test the quality of our model. We have made some predictions. Let's compare
# the model's prediction with actual target values for the test set. By following this method,
# we are going to calculate the accuracy of our model.
# measuring the accuracy of the model, using the actual data and the predicted results
accuracy = accuracy_score(target_test, pred)

# Displaying the accuracy of the model
print("\nModel Accuracy = ",accuracy*100,"%")

# Now suppose we want to predict for the conditions:
# Outlook = Rain (Rain is represented as 1 in the Outlook class)
```

```
# Temperature = Mild (Mild is represented as 2 in the Temperature class)
# Humidity = High (High is represented as 0 in the Humidity class)
# Wind = Weak (Weak is represented as 1 in the Wind class)
# Should we play (1) or not (0) ? According to our data set, given these features play should be 1
answer = model.predict([[1,2,0,1]])

if answer == 1:
    print("\nPlay")
elif answer == 0:
    print("\nNo Play")

#Awesome! Our model is giving an accuracy of 80%. This is not bad with a simple implementation.
# You can create random test datasets and test the model to get know how well the trained Gaussian Naive Bayes model
# is performing.
```

Activity/Assessment:

1. Simulate the program and analyze the given data set from table 1 and try to have different combinations of predictors for outlook, humidity, temperature and windy to know when to play golf or not.

APPENDICES

Sample Programs

Input/Output/Assignment Statements

```
x = int(input('enter an integer number: '))
print (x)
x = 1.5
print(x)
x = 'a'
print (x)
x = 'hello world!'
print (x)
```

Arithmetic Operations

```
a = int(input('enter 1st number: '))
b = int(input('enter 2nd number: '))
c = a + b
print ('the sum of 2 numbers is: ',c)
c = a - b
print ('the difference of 2 numbers is: ',c)
c = a * b
print ('the product of 2 numbers is: ',c)
c = a / b
print ('the quotient of 2 numbers is: ',c)
c = a // b
print ('the whole number of quotient of 2 numbers is: ',c)
c = a % b
print ('the remainder part of the quotient of 2 numbers is: ',c)
c = a ** b
print ('first number raised to the power of the second number is: ',c)
```

Conditional Statement

```
num = int(input('enter a number: '))
if num > 0:
    print('positive number')
elif (num < 0):
    print ('negative number')
else: print ('zero')
```

Repetition/Looping Statements:

1)

```
num = int(input('enter a number: '))
for i in range(1,num):
    print(i)
print()
for i in range(1,num+1):
```

```
    print(i)
print()
for i in range(1,num,2):
    print(i)

2)
num = int(input('enter a number: '))
i = 1
while i <= num:
    print(i)
    i += 1
```

List/Tuple/Dictionary

List

```
lst = []
x = int(input('how many values to be inserted in the list?:'))
for i in range(1,x+1):
    a = input('value: ')
    lst.append(a)
for i in range(len(lst)):
    print(lst[i])
print(lst)
```

Tuple

```
tpl = () # tuple variable
lst = [] # list variable
x = int(input('how many values to be inserted in the tuple?:'))
for i in range(1,x+1):
    a = input('value: ')
    lst.append(a) # inserting value in a list lst
tpl = tuple(lst) # converting the lst to tuple
for i in range(len(tpl)):
    print(tpl[i])
print(tpl)
```

Dictionary

```
dic = {'Name': '',  'Age':0}
name = input('name: ')
age = int(input('age: '))
dic['Name'] = name
dic['Age'] = age
print()
print('Name: ',dic['Name'] + '\tAge: ',dic['Age'])
```

Functions

1)
```
def info(age, name):
    print('name: ', name)
    print('age: ', age)

info(name = 'ely', age = 18)
```

2)
```
def main():
    num = (int)(input('enter a no.: '))
    factorial(num)

def factorial(no):
    fact = 1
    i = no
    while(i > 1):
        fact *= i
        i = i - 1
    print('factorial: ', fact)
main()
```

3)
```
lst = [1,2,3,4,5]
def output(mylist):
    for i in mylist:
        print(i)
    print()
def out(listing):
    for i in listing:
        print(i)

output(lst)
out(lst)
```

4)
```
grades=[0,0,0,0,0,0]
def inputs():
    namne = input('enter a name: ')
    for i in range(0,6):
        g = (float)(input('enter a grade: '))
        grades.append(g)
    ave = average(grades)
```

```
    print('average: ', ave)
    remarks(ave)

def average(grd):
    sum = 0
    for i in grades:
        sum += i
    ave = sum / 6
    return(ave)

def remarks(ave):
    if(ave <=3.12):
        rem = 'Passed'
    else:
        rem = 'Failed'
    print('Reamarks:',rem)

inputs()
```

Lambda

1)

```
x = 5; y = 10
sum = lambda x, y: x + y
diff = lambda x, y: x - y;

print('sum: ',sum(x,y))
print('difference: ',diff(x,y))
```

2)

```
num1 = int(input('First No.: '))
num2 = int(input('Second No.: '))
sum = lambda num1, num2: num1 + num2
diff = lambda num1, num2: num1 - num2
prod = lambda num1, num2: num1 * num2
quo = lambda num1, num2: num1 / num2
mod = lambda num1, num2: num1 % num2
div = lambda num1, num2: num1 // num2
expo = lambda num1, num2: num1 ** num2
print('Sum: ',sum(num1,num2))
print('Difference: ',diff(num1,num2))
print('Product: ',prod(num1,num2))
print('Quotient: ',quo(num1,num2))
print('Remainder: ',mod(num1,num2))
print('Whole Number in Quotient: ',div(num1,num2))
print('Exponent: ',expo(num1,num2))
```

Exception Handling

1)
```
def KelvinToFahrenheit(Temperature):
   assert (Temperature >= 0),"Colder than absolute zero!"
   return ((Temperature-273)*1.8)+32
print (KelvinToFahrenheit(273))
print (int(KelvinToFahrenheit(505.78)))
print (KelvinToFahrenheit(-5))
```

2)
```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print ("Error: can\'t find file or read data")
else:
   print ("Written content in the file successfully")
   fh.close()
```

3)
```
try:
   fh = open("testfile", "r")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print ("Error: can\'t find file or read data")
else:
   print ("Written content in the file successfully")
```

4)
```
# Define a function here.
def temp_convert(var):
   try:
      return int(var)
   except ValueError, Argument:
      print "The argument does not contain numbers\n", Argument

# Call above function here.
temp_convert("xyz");
```

Classes

1)
```
class Employee:
    def __init__(self,name,salary):
        self.name = name
        self.salary = salary
   def outputs(self):
```

```
        print('employee name: ',self.name)
        print('salary: ',self.salary)

def inputs():
        name = input('employee name: ')
        salary = (float)(input('input salary: '))
        emp = Employee(name,salary)
        emp.outputs()

inputs()
```

2)

```
class Employee:
    def __init__(self,name,gross):
        self.name = name
        self.gross = gross
    def outputs(self):
        print('employee name: ',self.name)
        print('gross pay: ',self.gross)

    class GrossPay:
        def __init__(self,rate,hrs):
            self.rate = rate
            self.hrs = hrs
        def compute(self):
            return (self.rate * self.hrs)

def inputs():
        neym = input('employee name: ')
        r8 = (float)(input('input rate per hour: '))
        hr = int(input('input hours worked: '))
        #emp = Employee(r8,hr)
        emp1 = Employee.GrossPay(r8,hr)
        #gross = emp1.compute()
        gross = Employee.GrossPay.compute(Employee.GrossPay(r8,hr))
        emp = Employee(neym,gross)
        emp.outputs()

inputs()
```

3)

```
class Employee:
    def __init__(self,rate,hrs):
        self.rate = rate
        self.hrs = hrs
```

```
    def __init__(self,name,gross):
        self.name = name
        self.gross = gross
    def outputs(self):
        print('employee name: ',self.name)
        print('gross pay: ',self.gross)

    class GrossPay:
        def __init__(self,rate,hrs):
            self.rate = rate
            self.hrs = hrs
        def compute(self):
            return (self.rate * self.hrs)

def inputs():
        neym = input('employee name: ')
        r8 = (float)(input('input rate per hour: '))
        hr = int(input('input hours worked: '))
        emp = Employee(r8,hr)
        emp1 = Employee.GrossPay(r8,hr)
        gross = emp1.compute()
        gross = emp.GrossPay.compute(emp.GrossPay(r8,hr))
        emp = Employee(neym,gross)
        emp.outputs()

inputs()

4)
class Sales:
    total = 0
    def __init__(self,sales):
        self.sales = sales
    def compute(self):
        Sales.total += self.sales
    def outputs():
        print('total sales: ',Sales.total)
def inputs():
    tot = 0
    #for j in range(0,3):

    for i in range(0,4):
        from salesman import Salesman
        sno = int(input('salesman no.: '))
        sna = input('salesman name: ')
```

```
        amt = float(input('sales amount: '))
        seyls = Sales(amt)
        tot = seyls.compute()
        sman = Salesman(sno,sna)

        seyls = Sales.outputs()
        sman.display()
        tot = 0
        #import salesman
inputs()

5)
class Salesman:
    def __init__(self,snum,sname):
        self.snum = snum
        self.sname = sname
    def display(self):
        print('salesman number: ', self.snum)
        print('salesman name: ',self.sname)
def inputs():
    sno = int(input('salesman no.: '))
    sna = input('salesman name: ')
    sman = Salesman(sno,sna)
    sman.display()

inputs()

6)
class Parent:        # define parent class
   parentAttr = 100
   def __init__(self):
      print "Calling parent constructor"

   def parentMethod(self):
      print 'Calling parent method'

   def setAttr(self, attr):
      Parent.parentAttr = attr

   def getAttr(self):
      print ("Parent attribute :", Parent.parentAttr)

class Child(Parent): # define child class
   def __init__(self):
      print ("Calling child constructor")
```

```
   def childMethod(self):
      print( 'Calling child method')

c = Child()          # instance of child
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.getAttr()          # again call parent's method

7)
class Parent:        # define parent class
   def myMethod(self):
      print( 'Calling parent method')

class Child(Parent): # define child class
   def myMethod(self):
      print ('Calling child method')

c = Child()          # instance of child
c.myMethod()         # child calls overridden method
```

GUI (Tkinter)

1) **Buttons**

```
import Tkinter
import tkMessageBox

top = Tkinter.Tk()

def helloCallBack():
   tkMessageBox.showinfo( "Hello Python", "Hello World")

B = Tkinter.Button(top, text ="Hello", command = helloCallBack)

B.pack()
top.mainloop()
```

2) **Checkbuttons**

```
from Tkinter import *
import tkMessageBox
import Tkinter

top = Tkinter.Tk()
CheckVar1 = IntVar()
```

```
CheckVar2 = IntVar()
C1 = Checkbutton(top, text = "Music", variable = CheckVar1, \
            onvalue = 1, offvalue = 0, height=5, \
            width = 20)
C2 = Checkbutton(top, text = "Video", variable = CheckVar2, \
            onvalue = 1, offvalue = 0, height=5, \
            width = 20)
C1.pack()
C2.pack()
top.mainloop()
```

3) **Entry**

```
from Tkinter import *

top = Tk()
L1 = Label(top, text="User Name")
L1.pack( side = LEFT)
E1 = Entry(top, bd =5)
E1.pack(side = RIGHT)

top.mainloop()
```

4) **Label**

```
from Tkinter import *

root = Tk()
var = StringVar()
label = Label( root, textvariable=var, relief=RAISED )

var.set("Hey!? How are you doing?")
label.pack()
root.mainloop()
```

5) **Listbox**

```
from Tkinter import *
import tkMessageBox
import Tkinter

top = Tk()

Lb1 = Listbox(top)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.insert(4, "PHP")
```

```
Lb1.insert(5, "JSP")
Lb1.insert(6, "Ruby")

Lb1.pack()
top.mainloop()
```

6) **Message**

```
from Tkinter import *

root = Tk()
var = StringVar()
label = Message( root, textvariable=var, relief=RAISED )

var.set("Hey!? How are you doing?")
label.pack()
root.mainloop()
```

7) **Radiobutton**

```
from Tkinter import *

def sel():
   selection = "You selected the option " + str(var.get())
   label.config(text = selection)

root = Tk()
var = IntVar()
R1 = Radiobutton(root, text="Option 1", variable=var, value=1, command=sel)
R1.pack( anchor = W )

R2 = Radiobutton(root, text="Option 2", variable=var, value=2, command=sel)
R2.pack( anchor = W )

R3 = Radiobutton(root, text="Option 3", variable=var, value=3, command=sel)
R3.pack( anchor = W)

label = Label(root)
label.pack()
root.mainloop()
```

8) **Text**

```
from Tkinter import *

def onclick():
   pass
```

```
root = Tk()
text = Text(root)
text.insert(INSERT, "Hello.....")
text.insert(END, "Bye Bye.....")
text.pack()

text.tag_add("here", "1.0", "1.4")
text.tag_add("start", "1.8", "1.13")
text.tag_config("here", background="yellow", foreground="blue")
text.tag_config("start", background="black", foreground="green")
root.mainloop()
```

9) **tkMessageBox**

```
import Tkinter
import tkMessageBox

top = Tkinter.Tk()
def hello():
   tkMessageBox.showinfo("Say Hello", "Hello World")

B1 = Tkinter.Button(top, text = "Say Hello", command = hello)
B1.pack()

top.mainloop()
```

10) **Frame**

```
from Tkinter import *

def donothing():
   filewin = Toplevel(root)
   button = Button(filewin, text="Do nothing button")
   button.pack()

root = Tk()
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="New", command=donothing)
filemenu.add_command(label="Open", command=donothing)
filemenu.add_command(label="Save", command=donothing)
filemenu.add_command(label="Save as...", command=donothing)
filemenu.add_command(label="Close", command=donothing)

filemenu.add_separator()

filemenu.add_command(label="Exit", command=root.quit)
```

```
menubar.add_cascade(label="File", menu=filemenu)
editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label="Undo", command=donothing)

editmenu.add_separator()

editmenu.add_command(label="Cut", command=donothing)
editmenu.add_command(label="Copy", command=donothing)
editmenu.add_command(label="Paste", command=donothing)
editmenu.add_command(label="Delete", command=donothing)
editmenu.add_command(label="Select All", command=donothing)

menubar.add_cascade(label="Edit", menu=editmenu)
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="Help Index", command=donothing)
helpmenu.add_command(label="About...", command=donothing)
menubar.add_cascade(label="Help", menu=helpmenu)

root.config(menu=menubar)
root.mainloop()
```

11) **ScrollBar**

```
from Tkinter import *

root = Tk()
scrollbar = Scrollbar(root)
scrollbar.pack( side = RIGHT, fill = Y )

mylist = Listbox(root, yscrollcommand = scrollbar.set )
for line in range(100):
   mylist.insert(END, "This is line number " + str(line))

mylist.pack( side = LEFT, fill = BOTH )
scrollbar.config( command = mylist.yview )

mainloop()
```

12) **PanedWindow**

```
from Tkinter import *

m1 = PanedWindow()
m1.pack(fill=BOTH, expand=1)

left = Label(m1, text="left pane")
m1.add(left)
```

```
m2 = PanedWindow(m1, orient=VERTICAL)
m1.add(m2)

top = Label(m2, text="top pane")
m2.add(top)

bottom = Label(m2, text="bottom pane")
m2.add(bottom)

mainloop()
```

13) **LabelFrame**

```
from Tkinter import *

root = Tk()

labelframe = LabelFrame(root, text="This is a LabelFrame")
labelframe.pack(fill="both", expand="yes")

left = Label(labelframe, text="Inside the LabelFrame")
left.pack()

root.mainloop()
```

14) **MenuButton**

```
from Tkinter import *
import tkMessageBox
import Tkinter

top = Tk()

mb=  Menubutton ( top, text="condiments", relief=RAISED )
mb.grid()
mb.menu =  Menu ( mb, tearoff = 0 )
mb["menu"] =  mb.menu

mayoVar = IntVar()
ketchVar = IntVar()

mb.menu.add_checkbutton ( label="mayo", variable=mayoVar )
mb.menu.add_checkbutton ( label="ketchup", variable=ketchVar )

mb.pack()
top.mainloop()
```

Modules

1)
```
import func3

lst = [2, 5,'hello', 4.7]
func3.output(lst)
func3.out(lst)
```

2)
```
from func3 import out

lst = [2, 5,'hello', 4.7]
out(lst)
```

3)
```
import funct2
num = (int) (input('enter a no.: '))
from funct2 import factorial
factorial(num)
```

4)
```
import employee

for i in range(0,10):
    employee.inputs()
```

5)
```
import class1
class1.inputs()

from class1 import Employee
name = input('new employee name: ')
salary = (float)(input('new input salary: '))
emp = Employee(name,salary)
emp.outputs()
```

6)
```
import class2
def inputs():
    for i in range(0,2):
        neym = input('employee name: ')
        r8 = (float)(input('input rate per hour: '))
        hr = int(input('input hours worked: '))
        employee = class2.Employee(r8,hr)
```

```
        compute(employee,neym)
def compute(employ,name,rate,hour):
        gross = employ.GrossPay.compute(employ.GrossPay(rate,hour))
        employ = class2.Employee(name,gross)
        employ.outputs()

inputs()
```

Files

1)

```
name = "prof. x"
age = 58
value = name + "\t" + str(age)
file = open("sample.txt","w")
file.write(value)
for i in range(0,3):
    name = input('name: ')
    age = int(input('age: '))
    value = name + "\t" + str(age)
    file.write(value)
file.close()
file = open("sample.txt","r")
lines = file.readlines()
for value in lines:
    print(value)
file.close()
```

2)

```
with open("sample.txt","r") as file:
    print(file.read())
```

3)

```
file = open("sample.txt","r")
lines = file.readlines()
for value in lines:
    print(value)
file.close()

file = open("sample.txt","r")
```

4)

```
name = "logan"
age = 33

file = open("record.txt","w")
```

```
value = name + "\t" + str(age)
file.write(value)
file.close()
with open("record.txt","r") as file:
    print(file.read())

5)
name = "x-men"
found = 0
with open("sample.txt","r") as file:
    for line in file:
        if line.find(name) == 0:
            found = 1
            break
if found == 1:
    print("found")
else: print("not found")
file.close()

6)
import os

class Employee:

    def __init__(self,name,rate,hrs,gross):
        self.name = name
        self.rate = rate
        self.hrs = hrs
        self.gross = gross

    def write_file1(self):
        file = open("employee.txt","a")
        emprec = self.name + "\t" + str(self.rate) + "\t" + \
                str(self.hrs) + "\t" + str(self.gross) +'\n'
        file.write(emprec)

    def write_file2(self):
        tempfile = open("temporary.txt","a")
        emprec = self.name + "\t" + str(self.rate) + "\t" + \
                str(self.hrs) + "\t" + str(self.gross) +'\n'
        tempfile.write(emprec)
        #tempfile.close()

    class GrossPay:
        def __init__(self,rate,hrs):
```

```
            self.rate = rate
            self.hrs = hrs
        def compute(self):
            return (self.rate * self.hrs)

def add_rec():
        ans = 'y'
        file = open("employee.txt","a")
        while ans == 'y':
            neym = input('employee name: ')
            r8 = (float)(input('input rate per hour: '))
            hr = int(input('input hours worked: '))
            emp1 = Employee.GrossPay(r8,hr)
            gross = emp1.compute()
            #gross = Employee.GrossPay.compute(Employee.GrossPay(r8,hr))
            emp = Employee(neym,r8,hr,gross)
            emp.write_file1()
            ans = input('input again?[y/n]: ')
        file.close()
        #disp_rec()
        #file.close()

def disp_rec():
    print('\n\n')
    print("Name\tRate\tHours\tGross\n")
    with open("employee.txt","r") as file:
        for line in file:
            print(line)
    file.close()

def edit_rec():
    edname = input('input employee name to be edited: ')
    tempfile = open("temporary.txt","a")
    file = open("employee.txt","r")
    for line in file:
        if line.find(edname) == 0:
            #from file6 import Employee
            neym = input('new employee name: ')
            r8 = (float)(input('new rate per hour: '))
            hr = int(input('new hours worked: '))
            emp1 = Employee.GrossPay(r8,hr)
            gross = emp1.compute()
            gross = Employee.GrossPay.compute(Employee.GrossPay(r8,hr))
            emp = Employee(neym,r8,hr,gross)
            emp.write_file2()
```

```
            #tempfile.write(emp.write_file2.emprec)
        else: tempfile.write(line)
    file.close()
    tempfile.close()
    os.remove("employee.txt")
    os.rename("temporary.txt","employee.txt")

    #from file6 import disp_rec
    #disp_rec()

def del_rec():
    edname = input('input employee name to be deleted: ')
    tempfile = open("temporary.txt","a")
    file = open("employee.txt","r")
    for line in file:
        if line.find(edname) != 0:
            tempfile.write(line)
    file.close()
    tempfile.close()
    os.remove("employee.txt")
    os.rename("temporary.txt","employee.txt")

    #from file6 import disp_rec
    #disp_rec()

def transaction():
    ans = 'Y'
    while ans != 'X':
        print("\t\t\t\t[A]dd\n")
        print("\t\t\t\t[E]dit\n")
        print("\t\t\t\t[D]elete\n")
        print("\t\t\t\tD[I]splay\n")
        print("\t\t\t\tE[X]it\n\n\n")
        ans = input("\tChoose your transaction: ")
        if ans == 'A':
            add_rec()
        elif ans == 'E':
            edit_rec()
        elif ans == 'D':
            del_rec()
        elif ans == 'I':
            disp_rec()
        else: print('End of Program!!!')

transaction()
```

7)
```
import os
def edit_rec():
    edname = input('input employee name to be edited: ')
    tempfile = open("temporary.txt","w")
    file = open("employee.txt","r")
    for line in file:
        if line.find(edname) == 0:
            from file6 import Employee
            neym = input('new employee name: ')
            r8 = (float)(input('new rate per hour: '))
            hr = int(input('new hours worked: '))
            emp1 = Employee.GrossPay(r8,hr)
            gross = emp1.compute()
            gross = Employee.GrossPay.compute(Employee.GrossPay(r8,hr))
            emp = Employee(neym,r8,hr,gross)
            emp.write_file2()
            tempfile.write(emp.write_file2.emprec)
        else: tempfile.write(line)
    file.close()
    tempfile.close()
    os.remove("employee.txt")
    os.rename("temporary.txt","employee.txt")

    from file6 import disp_rec
    disp_rec()

edit_rec()
#os.remove("record.txt")
```

8)
```
import os

val1 = "Valid"
inval = "Invalid"
found = 0

file2 = open("input.txt","r")
file3 = open("output.txt","w")
file3.write("Token\t\t\tValidity\n\n")
for line1 in file2:
    file1 = open("other_tbl.txt","r")
    for line2 in file1:
        if line2.find(line1) == 0:
```

```
            found = 1
            break
    if found == 1:
        file3.write(line1 + "\t\t\t" + val1  + "\n\n")
    else:
        file3.write(line1 + "\t\t\t" + inval  + "\n\n")
    file1.close()
    found = 0

file2.close()
file3.close()
print('end of analysis!')
x = input()
```

Input File: input.txt

```
string
GET
TIMES
+=
//
THEN
else
```

other_tbl.txt

```
GET
VIEW
BOOLEAN
CHARACTER
INTEGER
REAL
STRING
if
then
else
LOOP
TIMES
DO
NOT
AND
OR
+
-
*
/
%
```

^
>
>=
<
<=
=
<>
:=
(
)
;
{
}
'
"
</
/>
<//

Output File: **output.txt**

Token	Validity
string	Invalid
GET	Valid
TIMES	Valid
+=	Invalid
//	Invalid
THEN	Invalid
else	Valid

Databases

1)

```
import sqlite3 as sq

conn = sq.connect("test.db")
#conn.execute("alter table sample_tbl(id int key not null," \
#               "name text, age int)")
cursor = conn.cursor()
print("table created successfully!!!")
ans = 'y'
```

```
while ans == 'y':
    pid = int(input('id no.: '))
    name = input('name: ')
    age = int(input('age: '))
    cursor.execute('insert into sample_tbl(id,name,age) values (?,?,?)',(pid,name,age))
    conn.commit()
    ans = input('new record[y/n]?: ')
conn.close()

conn = sq.connect("test.db")
cursor = conn.execute("select * from sample_tbl")
for column in cursor:
    print('id no.: ',column[0])
    print('name: ',column[1])
    print('age: ',column[2])
    print()
conn.close()
```

2)

```
import sqlite3 as sql

rem = ""; sno=0; sna=''; ave=0;
conn = sql.connect("student.db")
cursor = conn.cursor()
class Students:
    def __init__(self,sno,sna,ave,rem):
        self.sno = sno
        self.sna = sna
        self.ave = ave
        self.rem = rem
    def add_rec(self,cmd):
        cursor.execute(cmd,(self.sno,self.sna,self.ave,self.rem))
        conn.commit()

    class Compute:
        def __init__(self,midterm,final):
            self.midterm = midterm
            self.final = final
        def comp_ave(self):
            return ((self.midterm + self.final) / 2)

def main():
    #conn = sql.connect("student.db")
    #cursor = conn.cursor()
```

```
    #cmd = "create table tbl_stud(studno int primary key, studname text," \
    #           "ave float, rem text)"
    #cursor.execute(cmd)
    conn.commit()
    ans = 'y'
    while ans == 'y':
        sno = input('student no.: ')
        sna = input('student name: ')
        mid = float(input('midterm grade: '))
        fin = float(input('final grade: '))
        stud = Students.Compute(mid,fin)
        ave = stud.comp_ave()
        if ave <= 3.12:
            rem = 'passed'
        else: rem = 'failed'
        stud = Students(sno,sna,ave,rem)
        cmd = 'insert into tbl_stud(studno,studname,ave,rem) values(?,?,?,?)'
        stud.add_rec(cmd)
        ans = input('input again[y/n]?: ')
    conn.close()
    view_rec()

def view_rec():
    conn = sql.connect("student.db")
    cursor = conn.cursor()
    cursor = conn.execute('select * from tbl_stud')
    for record in cursor:
        print('student no.: ',record[0])
        print('student name: ',record[1])
        print('average: ',record[2])
        print('remarks: ',record[3])
        print()

3)
import sqlite3 as sq
conn = sq.connect("test.db")
cursor = conn.cursor()
def main():
    found = 0

    num = int(input('id no.: '))

    cursor.execute('select * from sample_tbl where id = ?',(num,))

    rows = cursor.fetchall()
```

```
    ctr = len(rows)
    for row in rows:
        if row in rows:
            print('record found!!!')
            print('id no.: ',row[0])
            print('name: ',row[1])
            print('age: ',row[2])
            edit(num)
            break
        else:
           found += 1

    if found == ctr:
        print('record not exists!')
    conn.close()

def edit(pid):
    name = input('name: ')
    age = int(input('age: '))
    cmd = 'update sample_tbl set name = ?, age = ? where id = ?'
    data = (name,age,pid)
    cursor.execute(cmd,data)
    conn.commit()
main()
```

4)

```
import sqlite3 as sq
conn = sq.connect("test.db")
cursor = conn.cursor()
def main():
    found = 0

    num = int(input('id no.: '))

    cursor.execute('select * from sample_tbl where id = ?',(num,))
    rows = cursor.fetchall()
    ctr = len(rows)
    for row in rows:
        if row in rows:
            print('record found!!!')
            print('id no.: ',row[0])
            print('name: ',row[1])
            print('age: ',row[2])
            delete(num)
            break
```

```
        else:
          found += 1

    if found == ctr:
        print('record not exists!')
    conn.close()

def delete(pid):
    cursor.execute('delete from sample_tbl where id = ?',(pid,))
    conn.commit()

main()
```

5)

```
#main program
import sqlite3 as sql
import os

conn = sql.connect("midterm.db")
cursor = conn.cursor()
tot = 50; iskor = 0.00; grd = 0.00
class Students:
    def __init__(self, sno, sna, score, grd):
        self.sno = sno
        self.sna = sna
        self.score = score
        self.grd = grd
    def save_rec(self, cmd):
        cursor.execute (cmd, (self.sno, self.sna, self.score, self.grd))
        conn.commit()
    class Compute:
        def __init__(self, score):
            self.score = score
        def comp_grd(self):
            return(float(5 - ((self.score / tot) * 4)))

def main():
    #cmd = 'drop table tbl_midterm'
    #cursor.execute(cmd)
    #cmd = "create table tbl_midterm (studno text primary key, studna text," \
    #            "score int, grade float)"
    #cursor.execute(cmd)
    #conn.commit()
    snum = input("Student No.: ")
    sname = input ("Student Name: ")
```

```
    print('\nType the missing code in the given program.')
    print('After typing, double check your answer before pressing the ENTER key.')
    print('Be sure that the code supplied is with the compliance of the COBOL rules.')
    print('Single space only between character, word, statement or syntax.')
    c = input('Press any key to start the midterm exam.')
    import question
    iskor = question.display()
    student = Students.Compute(iskor)
    grd = float(student.comp_grd())
    student = Students(snum, sname,iskor,grd)
    cmd = 'insert into tbl_midterm (studno, studna, score, grade) values (?,?,?,?)'
    student.save_rec(cmd)
    conn.close()
    view_score()
    #c = input('')

def view_score():
    conn = sql.connect("midterm.db")
    cursor = conn.cursor()
    cursor = conn.execute('select * from tbl_midterm')
    print('')
    for record in cursor:
        print('Student No.: ', record[0])
        print('Student Name: ', record[1])
        print('Score: ', record[2])
        print('Grade: ', record[3])
        #print('');
        c = input('Press ENTER key')
main()

#connects the main program
import sqlite3 as sql
import os
import random
import time

score = 0; ch = ""
con = sql.connect("answers.db")
cursor = con.cursor()
#cmd = 'drop table tbl_answer'
#cursor.execute(cmd)
#cmd = 'create table tbl_answer (num int primary key, ans text)'
#cursor.execute(cmd)
def main():
```

```
    for i in range(1,51):
        ch = ""
        print('Answer: \t')
        ans = input(' ')
        cursor.execute("insert into tbl_answer (num, ans) values (?,?)", (i,ans))
        con.commit()
    con.close()
    #display()

def display():

    score = 0; ch = ''
    con = sql.connect('answers.db')
    cursor = con.cursor()
    #cmd = 'create table tbl_quest (num int primary key, tanong text, choices text, ans text)'
    #cursor.execute(cmd)

    #cursor = con.cursor()
    cursor = con.execute('select * from tbl_answer')
    for record in cursor:
        os.system('cls')
        print ("(",record[0],")")
        print('Please type the missing code:')
        ch = input('')
        print('\n\nPlease wait for the next item.')
        time.sleep(2)
        if (ch == record[1]):
            score = score + 1

    con.close()
```