Object Oriented Python

tutorialspoint

SIMPLY EASY LEARNING

About the Tutorial

Python has been an object-oriented language since it existed. In this tutorial we will try to get in-depth features of OOPS in Python programming.

Audience

This tutorial has been prepared for the beginners and intermediate to help them understand the Python Oops features and concepts through programming.

Prerequisites

Understanding on basic of Python programming language will help to understand and learn quickly. If you are new to programming, it is recommended to first go through "Python for beginners" tutorials.

Table of Contents

About the Tutorial ii

Audience ii

Prerequisites ii

OOP IN PYTHON – INTRODUCTION 1

Language Programming Classification Scheme 1

What is Object Oriented Programming? 2

Why to Choose Object-oriented programming? 2

Procedural vs. Object Oriented Programming 2

Principles of Object Oriented Programming 3

Object-Oriented Python 5

Modules vs. Classes and Objects 5

OOP IN PYTHON – ENVIRONMENT SETUP 8

Prerequisites and Toolkits 8

Installing Python 8

Choosing an IDE 10

Pycharm 10

Komodo IDE 11

Eric Python IDE 12

Choosing a Text Editor 13

Atom Text Editor 13

Screenshot of Atom text 14

Sublime Text Editor 14

Notepad ++ 15

OOP IN PYTHON – DATA STRUCTURES 17

Lists 17

Accessing Items in Python List 18

Empty Objects 18

Tuples 19

Dictionary 21

Sets 24

OOP IN PYTHON – BUILDING BLOCKS 28

Class Bundles : Behavior and State 28

Creation and Instantiation 29

Instance Methods 30

Encapsulation 31

Init Constructor 33

Class Attributes 34

Working with Class and Instance Data 35

OOP IN PYTHON – OBJECT ORIENTED SHORTCUT 37

Python Built-in Functions 37

Default Arguments 42

OOP IN PYTHON – INHERITANCE AND POLYMORPHISM 44

Inheritance 44

Inheriting Attributes 44

Inheritance Examples 45

Polymorphism ("MANY SHAPES") 47

Overriding 48

Inheriting the Constructor 49

Multiple Inheritance and the Lookup Tree 50

Decorators, Static and Class Methods 54

OOP IN PYTHON –PYTHON DESIGN PATTERN 57

Overview **57**

Why is Design Pattern Important? **57**

Classification of Design Patterns **57**

Commonly used Design Patterns **58**

OOP IN PYTHON – ADVANCED FEATURES 60

Core Syntax in our Class design **60**

Inheriting From built-in types **61**

Naming Conventions **63**

OOP IN PYTHON – FILES AND STRINGS 65

Strings **66**

File I/O **71**

OOP IN PYTHON – EXCEPTION AND EXCEPTION CLASSES 72

Identifying Exception (Errors) **72**

Catching/Trapping Exception **73**

Raising Exceptions **75**

Creating Custom exception class **76**

OOP IN PYTHON – OBJECT SERIALIZATION 80

Pickle **80**

Methods **81**

Unpickling **82**

JSON **82**

YAML **85**

Installing YAML **85**

PDB – The Python Debugger **89**

Logging **91**

Benchmarking **93**

12. OOP IN PYTHON – PYTHON LIBRARIES 96

Requests: Python Requests Module 96

Making a GET Request 96

Making POST Requests 97

Pandas: Python Library Pandas 97

Indexing DataFrames 98

Pygame 99

Beautiful Soup: Web Scraping with Beautiful Soup 102

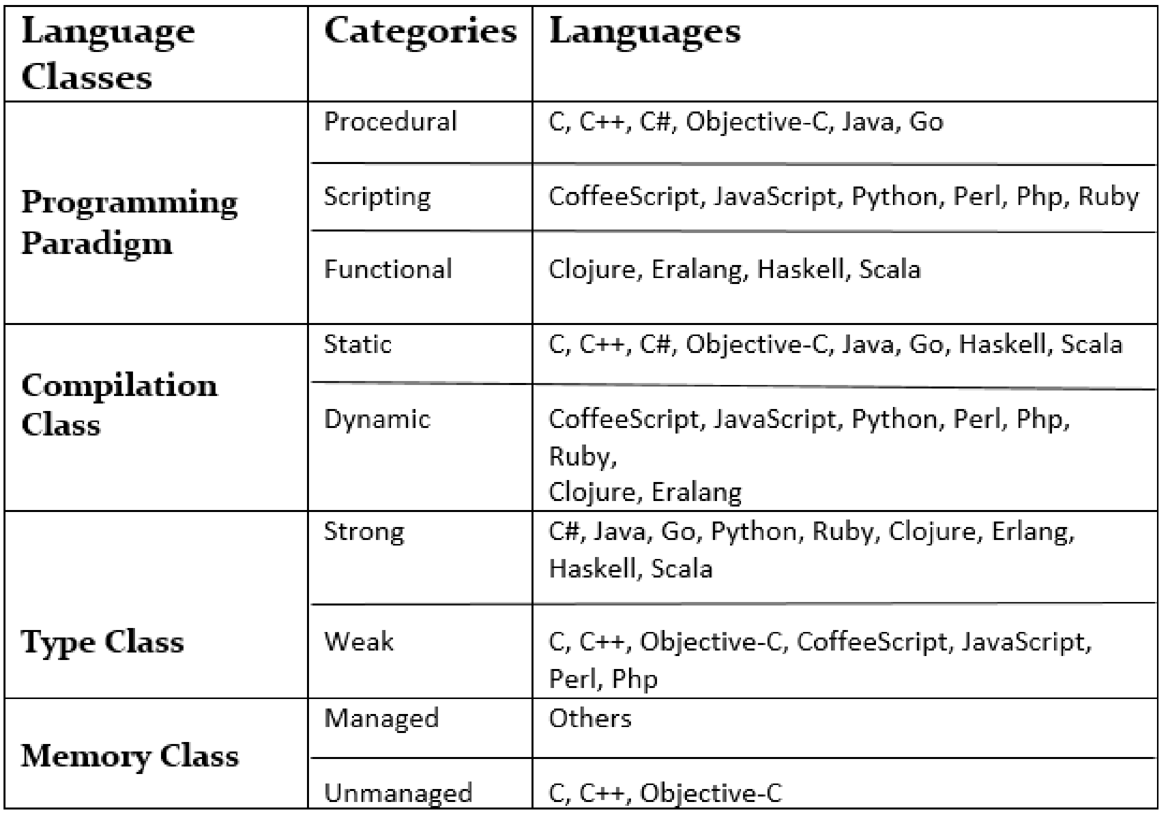
1. OOP in Python – Introduction

Programming languages are emerging constantly, and so are different methodologies. Object-oriented programming is one such methodology that has become quite popular over past few years.

This chapter talks about the features of Python programming language that makes it an object-oriented programming language.

Language Programming Classification Scheme

Python can be characterized under object-oriented programming methodologies. The following image shows the characteristics of various programming languages. Observe the features of Python that makes it object-oriented.



Language Classes	Categories	Languages
Programming Paradigm	Procedural	C, C++, C#, Objective-C, Java, Go
	Scripting	CoffeeScript, JavaScript, Python, Perl, Php, Ruby
	Functional	Clojure, Eralang, Haskell, Scala
Compilation Class	Static	C, C++, C#, Objective-C, Java, Go, Haskell, Scala
	Dynamic	CoffeeScript, JavaScript, Python, Perl, Php, Ruby, Clojure, Eralang
Type Class	Strong	C#, Java, Go, Python, Ruby, Clojure, Erlang, Haskell, Scala
	Weak	C, C++, Objective-C, CoffeeScript, JavaScript, Perl, Php
Memory Class	Managed	Others
	Unmanaged	C, C++, Objective-C

What is Object Oriented Programming?

Object Oriented means directed towards objects. In other words, it means functionally directed towards modelling objects. This is one of the many techniques used for modelling complex systems by describing a collection of interacting objects via their data and behavior.

Python, an Object Oriented programming (OOP), is a way of programming that focuses on using objects and classes to design and build applications.. Major pillars of Object Oriented Programming (OOP) are **Inheritance**, **Polymorphism**, **Abstraction**, ad **Encapsulation**.

Object Oriented Analysis(OOA) is the process of examining a problem, system or task and identifying the objects and interactions between them.

Why to Choose Object Oriented Programming?

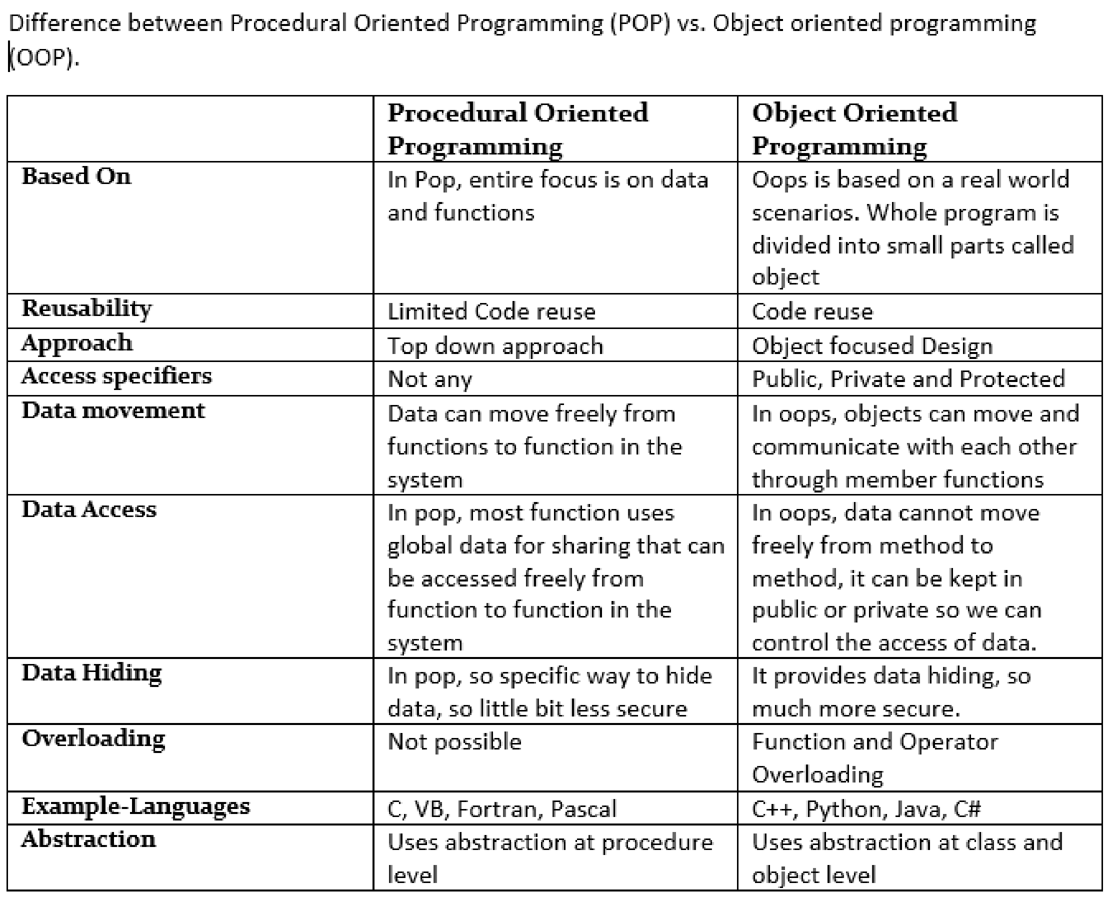
Python was designed with an object-oriented approach. OOP offers the following advantages:

- Provides a clear program structure, which makes it easy to map real world problems and their solutions.
- Facilitates easy maintenance and modification of existing code.
- Enhances program modularity because each object exists independently and new features can be added easily without disturbing the existing ones.
- Presents a good framework for code libraries where supplied components can be easily adapted and modified by the programmer.
- Imparts code reusability

Procedural vs. Object Oriented Programming

Procedural based programming is derived from structural programming based on the concepts of **functions/procedure/routines**. It is easy to access and change the data in procedural oriented programming. On the other hand, Object Oriented Programming (OOP) allows decomposition of a problem into a number of units called **objects** and then build the data and functions around these objects. It emphasis more on the data than procedure or functions. Also in OOP, data is hidden and cannot be accessed by external procedure.

The table in the following image shows the major differences between POP and OOP approach.

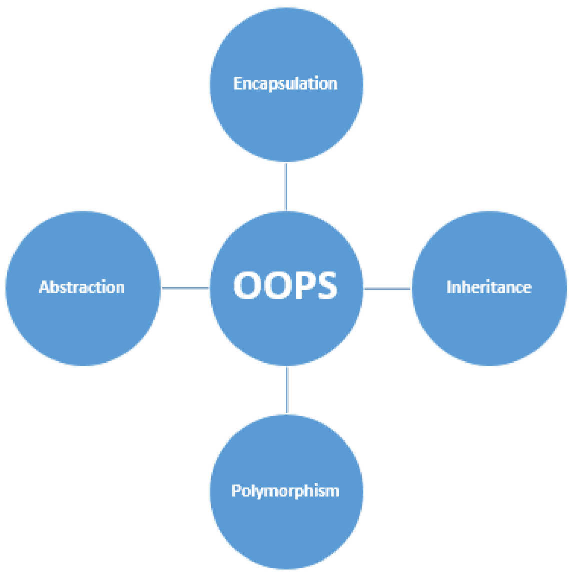


Difference between Procedural Oriented Programming (POP) vs. Object oriented programming (OOP).

	Procedural Oriented Programming	Object Oriented Programming
Based On	In Pop, entire focus is on data and functions	Oops is based on a real world scenarios. Whole program is divided into small parts called object
Reusability	Limited Code reuse	Code reuse
Approach	Top down approach	Object focused Design
Access specifiers	Not any	Public, Private and Protected
Data movement	Data can move freely from functions to function in the system	In oops, objects can move and communicate with each other through member functions
Data Access	In pop, most function uses global data for sharing that can be accessed freely from function to function in the system	In oops, data cannot move freely from method to method, it can be kept in public or private so we can control the access of data.
Data Hiding	In pop, so specific way to hide data, so little bit less secure	It provides data hiding, so much more secure.
Overloading	Not possible	Function and Operator Overloading
Example-Languages	C, VB, Fortran, Pascal	C++, Python, Java, C#
Abstraction	Uses abstraction at procedure level	Uses abstraction at class and object level

Principles of Object Oriented Programming

Object Oriented Programming (OOP) is based on the concept of **objects** rather than actions, and **data** rather than logic. In order for a programming language to be object-oriented, it should have a mechanism to enable working with classes and objects as well as the implementation and usage of the fundamental object-oriented principles and concepts namely inheritance, abstraction, encapsulation and polymorphism.



Four Pillars of Object-Oriented Programming

Let us understand each of the pillars of object-oriented programming in brief:

Encapsulation

This property hides unnecessary details and makes it easier to manage the program structure. Each object's implementation and state are hidden behind well-defined boundaries and that provides a clean and simple interface for working with them. One way to accomplish this is by making the data private.

Inheritance

Inheritance, also called generalization, allows us to capture a hierarchal relationship between classes and objects. For instance, a 'fruit' is a generalization of 'orange'. Inheritance is very useful from a code reuse perspective.

Abstraction

This property allows us to hide the details and expose only the essential features of a concept or object. For example, a person driving a scooter knows that on pressing a horn, sound is emitted, but he has no idea about how the sound is actually generated on pressing the horn.

Polymorphism

Poly-morphism means many forms. That is, a thing or action is present in different forms or ways. One good example of polymorphism is constructor overloading in classes.

Object-Oriented Python

The heart of Python programming is **object** and **OOP**, however you need not restrict yourself to use the OOP by organizing your code into classes. OOP adds to the whole design philosophy of Python and encourages a clean and pragmatic way to programming. OOP also enables in writing bigger and complex programs.

Modules vs. Classes and Objects

Modules are like "Dictionaries"

When working on Modules, note the following points:

- A Python module is a package to encapsulate reusable code.
- Modules reside in a folder with a **__init__.py** file on it.
- Modules contain functions and classes.
- Modules are imported using the **import** keyword.

Recall that a dictionary is a **key-value** pair. That means if you have a dictionary with a key **EmployeID** and you want to retrieve it, then you will have to use the following lines of code:

```
employee = {"EmployeID": "Employee Unique Identity!"}
print (employee ['EmployeID])
```

You will have to work on modules with the following process:

- A module is a Python file with some functions or variables in it.
- Import the file you need.
- Now, you can access the functions or variables in that module with the '**.**' **(dot)** Operator.

Consider a module named **employee.py** with a function in it called **employee**. The code of the function is given below:

```
# this goes in employee.py
def EmployeID():
     print ("Employee Unique Identity!")
```

Now import the module and then access the function **EmployeID**:

```
import employee
employee. EmployeID()
```

You can insert a variable in it named **Age**, as shown:

```
def EmployeID():
     print (“Employee Unique Identity!”)
# just a variable
Age = “Employee age is **”
```

Now, access that variable in the following way:

```
import employee
employee.EmployeID()
print(employee.Age)
```

Now, let’s compare this to dictionary:

```
Employee[‘EmployeID’]    # get EmployeID from employee
Employee.employeID()     # get employeID from the module
Employee.Age             # get access to variable
```

Notice that there is common pattern in Python:

- Take a **key = value** style container
- Get something out of it by the key’s name

When comparing module with a dictionary, both are similar, except with the following:

- In the case of the **dictionary**, the key is a string and the syntax is [key].
- In the case of the **module**, the key is an identifier, and the syntax is .key.

Classes are like Modules

Module is a specialized dictionary that can store Python code so you can get to it with the ‘**.**’ Operator. A class is a way to take a grouping of functions and data and place them inside a container so you can access them with the ‘**.**‘operator.

If you have to create a class similar to the employee module, you can do it using the following code:

```
class employee(object):
     def __init__(self):
            self. Age = “Employee Age is ##”
     def EmployeID(self):
            print (“This is just employee unique identity”)
```

Note: Classes are preferred over modules because you can reuse them as they are and without much interference. While with modules, you have only one with the entire program.

Objects are like Mini-imports

A class is like a **mini-module** and you can import in a similar way as you do for classes, using the concept called **instantiate**. Note that when you instantiate a class, you get an **object**.

You can instantiate an object, similar to calling a class like a function, as shown:

```
this_obj = employee()          # Instantiatethis_obj.EmployeID()            #
get EmployeId from the class

print(this_obj.Age)                  # get variable Age
```

You can do this in any of the following three ways:

```
# dictionary style

Employee[‘EmployeID’]

# module style

Employee.EmployeID()

Print(employee.Age)

# Class style

this_obj = employee()

this_obj.employeID()

Print(this_obj.Age)
```

2. OOP in Python – Environment Setup

This chapter will explain in detail about setting up the Python environment on your local computer.

Prerequisites and Toolkits

Before you proceed with learning further on Python, we suggest you to check whether the following prerequisites are met:

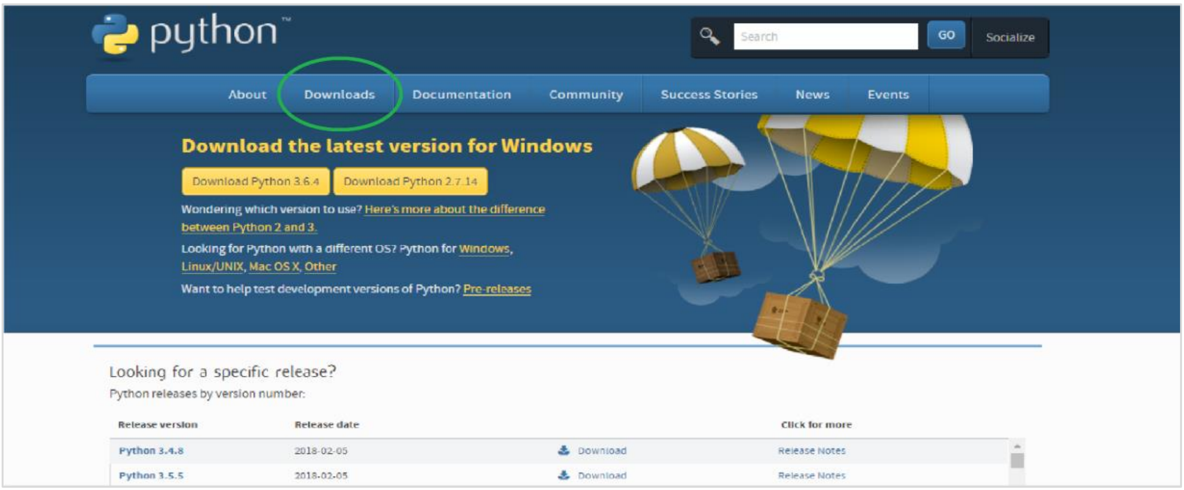
- Latest version of Python is installed on your computer
- An IDE or text editor is installed
- You have basic familiarity to write and debug in Python, that is you can do the following in Python:
  - Able to write and run Python programs.
  - Debug programs and diagnose errors.
  - Work with basic data types.
  - Write **for** loops, **while** loops, and **if** statements
  - Code **functions**

If you don't have any programming language experience, you can find lots of beginner tutorials in Python on TutorialsPoint.

Installing Python

The following steps show you in detail how to install Python on your local computer:

Step 1: Go to the official Python website https://www.Python.org/, click on the **Downloads** menu and choose the latest or any stable version of your choice.



Download the latest version for Windows

Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

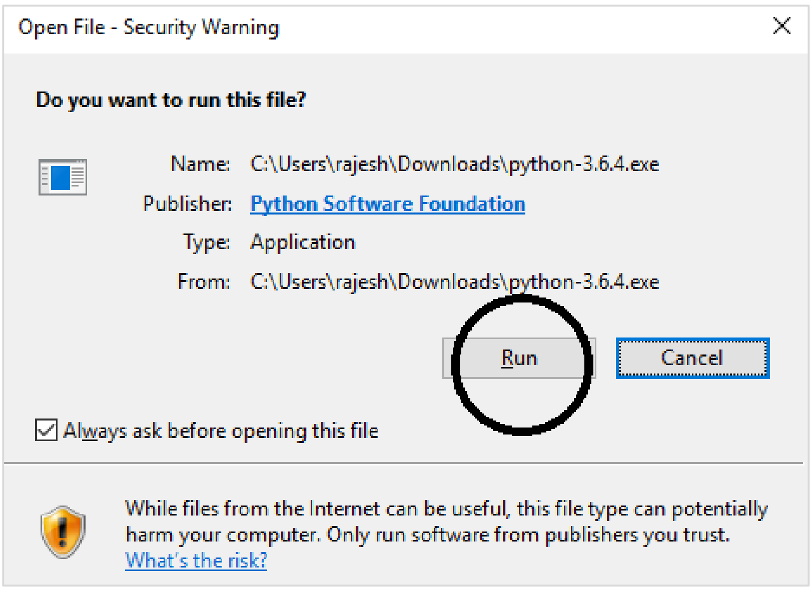
Want to help test development versions of Python? Pre-releases

Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.4.8	2018-02-05	Download	Release Notes
Python 3.5.5	2018-02-05	Download	Release Notes

Step 2: Save the Python installer exe file that you're downloading and once you have downloaded it, open it. Click on **Run** and choose **Next** option by default and finish the installation.



Open File - Security Warning

Do you want to run this file?

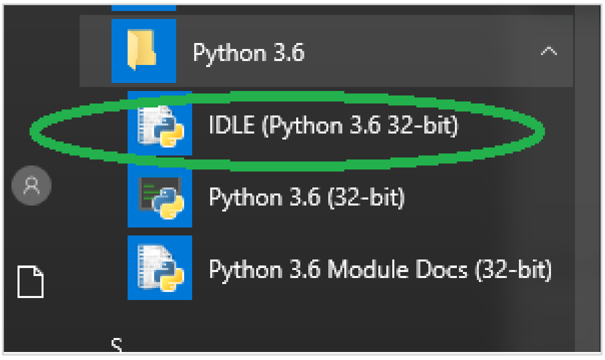
Name: C:\Users\rajesh\Downloads\python-3.6.4.exe
Publisher: Python Software Foundation
Type: Application
From: C:\Users\rajesh\Downloads\python-3.6.4.exe

Run Cancel

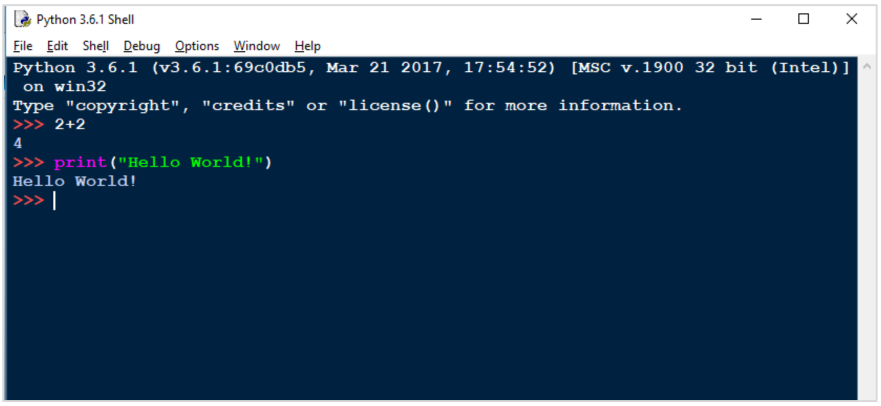
☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Step 3: After you have installed, you should now see the Python menu as shown in the image below. Start the program by choosing IDLE (Python GUI).



This will start the Python shell. Type in simple commands to check the installation.



```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> print("Hello World!")
Hello World!
>>> |
```