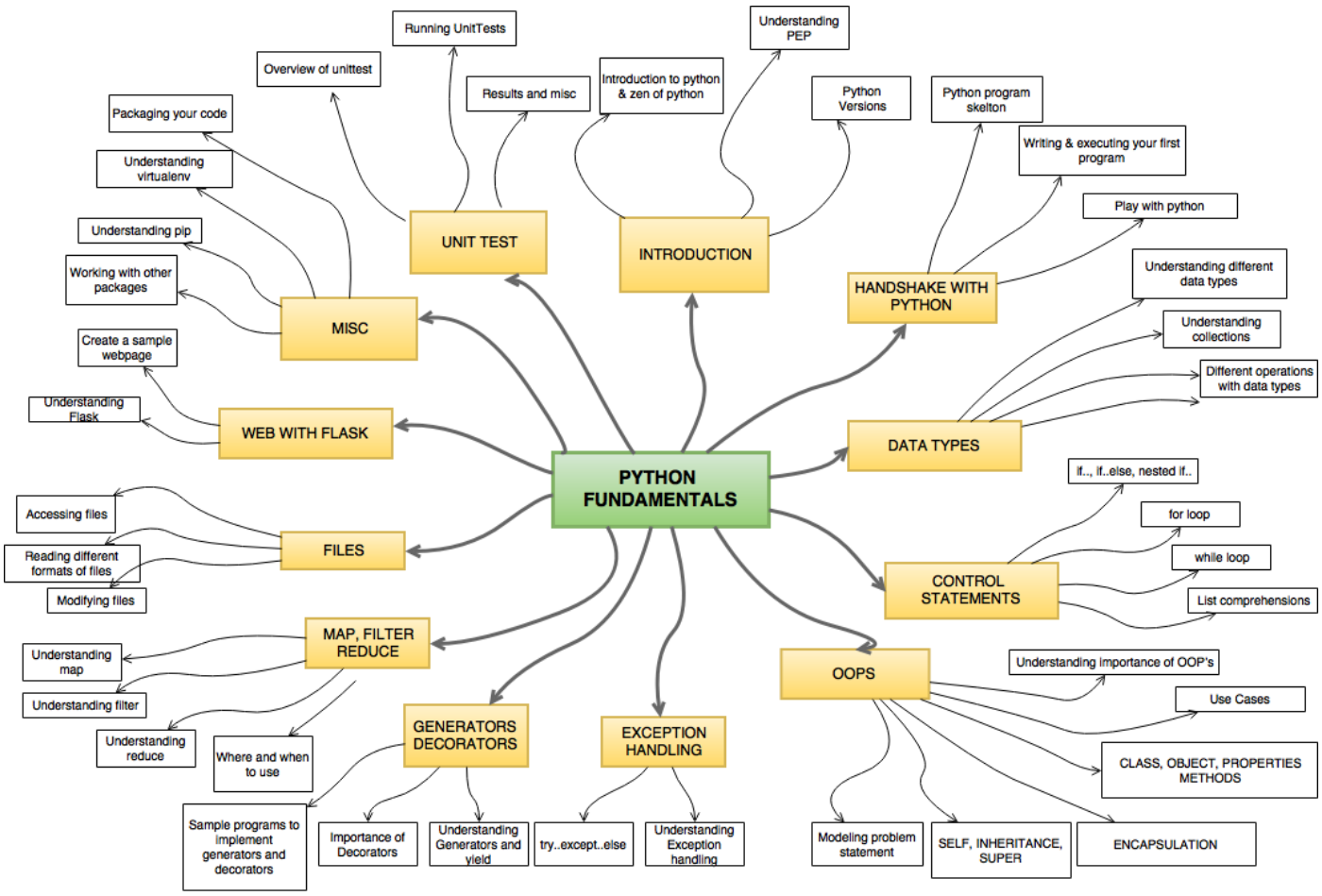


Source - https://wilsonmar.github.io/python-tutorials/

Python is an interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically-typed and garbage-collected programming language.

Python supports multiple programming paradigms, including **Procedural**, **Object Oriented** and **Functional programming language**

It was created by Guido van Rossum during 1985- 1990.

Using Python's pip to Manage Your Projects' Dependencies

- pip install is a command used in the Python programming language to install packages or libraries from the Python Package Index (PyPI). PyPI is a repository of software packages developed and shared by the Python community.

- Here's how you typically use pip install:

pip install package_name

Replace "package_name" with the name of the Python package you want to install. When you run this command, pip will download the specified package and its dependencies from PyPI and install them on your system.

Python Identifiers

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

Naming conventions for 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 (4 spaces or Tab key), which is rigidly enforced.

Example

```
if flag == True :
        print ('True')
else:
        print ('False')
```

Multi-Line Statements

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

Example

```
total = item_one + \
        item_two + \
        item_three
```

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

Quotation in Python

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.

Example

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Variables in Python

- Variable is a name attached to a particular object.
- In Python, variables need not be declared or defined in advance. To create a variable, you just assign it a value and then start using it.
- Assignment is done with a single equals sign (=):

```
>>> n = ‘Hello World’
```

- Python also allows chained assignment, which makes it possible to assign the same value to several variables simultaneously:

```
>>> a = b = c = 300
>>> print(a, b, c)
300 300 300
```

Variables in Python

- Python is a highly object-oriented language. Every item of data in a Python program is an object of a specific type or class.

Example

```
>>> print(300)
300
```

- When presented with the statement print(300), the interpreter does the following:
  - Creates an integer object
  - Gives it the value 300
  - Displays it to the console

- You can see that an integer object is created using the built-in type() function:

```
>>> type(300)
<class 'int'>
```

- A Python variable is a symbolic name that is a reference or pointer to an object. Once an object is assigned to a variable, you can refer to the object by that name. But the data itself is still contained within the object.

For example:

```
>>> n = 300
```

This assignment creates an integer object with the value 300 and assigns the variable **n** to point to that object.

- Now consider the following statement:

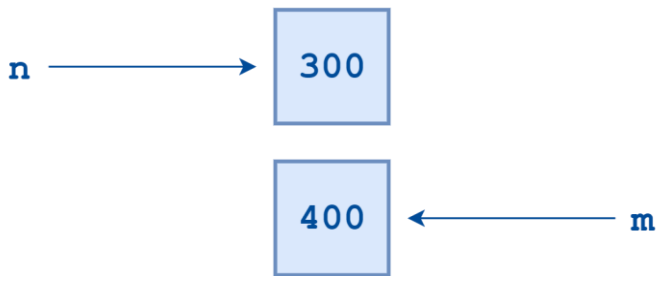
```
>>> m = n
```

Python does not create another object. It simply creates a new symbolic name or reference, m, which points to the same object that n points to.

- Next, suppose you do this:

>>> m = 400

Now Python creates a new integer object with the value 400, and m becomes a reference to it.



- An object's life begins when it is created, at which time at least one reference to it is created. During an object's lifetime, additional references to it may be created and references to it may be deleted as well. An object stays alive, as it were, so long as there is at least one reference to it.
- When the number of references to an object drops to zero, it is no longer accessible. At that point, its lifetime is over. Python will eventually notice that it is inaccessible and reclaim the allocated memory so it can be used for something else. In computer lingo, this process is referred to as garbage collection.

Variable Names

- Variable names in Python can be any length and can consist of uppercase and lowercase letters (A-Z, a-z), digits (0-9), and the underscore character (_). An additional restriction is that, although a variable name can contain digits, the first character of a variable name cannot be a digit.

Python Keywords

Python Keywords			
False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

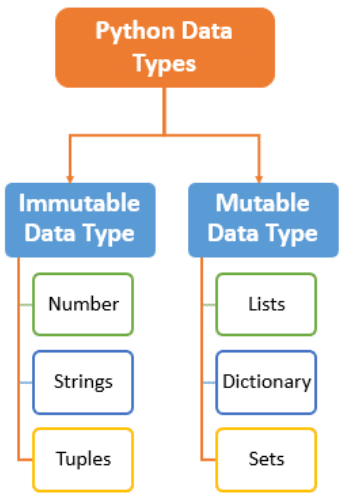
Python Data Types

Core Data Types in Python

Data type is the classification of the type of values that can be assigned to variables. In Python, we don't need to declare a variable with explicitly mentioning the data type, but it's still important to understand the different types of values that can be assigned to variables in Python.

Python data types are categorized as follows:

Mutable Data Types: Data types in python where the value assigned to a variable can be changed. Some mutable data types in Python include set, list, user-defined classes and dictionary.
Immutable Data Types: Data types in python where the value assigned to a variable cannot be changed. Some immutable data types in Python are int, decimal, float, tuple, bool, range and string.



Numbers: The number data type in Python is used to store numerical values. It is used to carry out normal mathematical operations.

Strings: Strings in Python are used to store textual information. They are used to carry out operations that perform positional ordering among items.

Lists: The list data type is the most generic Python data type. Lists can consist of a collection of mixed data types, stored by relative positions.

Tuples: Python Tuples are one among the immutable Python data types that can store values of mixed data types. They are basically a list that cannot be changed.

Sets: Sets in Python are a data type that can be considered as an unordered collection of data without any duplicate items.

Dictionaries: Dictionaries in Python can store multiple objects, but unlike lists, in dictionaries, the objects are stored by keys and not by positions.

Built-in Data Types in Python

- **Binary Types** – bytes, memory view, bytearray
- **Mapping Type** – dict
- **Numeric Type** – int, float, complex
- **Text Type** – str
- **Boolean Type** – bool
- **Set Types** – set, frozenset
- **Sequence Types** – list, range, tuple

How to check Data Type in Python

You can get the data type of any object by using the type() function:

```
n = 10
type(n)
<class 'int'>
```

Integers

In Python 3, there is effectively no limit to how long an integer value can be. Of course, it is constrained by the amount of memory your system has, as are all things, but beyond that an integer can be as long as you need it to be:

```
>>> print(123123123123123123123123123123123123123123123123 + 1)
123123123123123123123123123123123123123123123124
```

Floating-Point Numbers

The float type in Python designates a floating-point number. float values are specified with a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation:

```
>>> 4.2
4.2
>>> type(4.2)
<class 'float'>
>>> .4e7
4000000.0
>>> type(.4e7)
<class 'float'>
>>> 4.2e-4
0.00042
```

Strings

Strings are sequences of character data. The string type in Python is called str.

String literals may be delimited using either single or double quotes. All the characters between the opening delimiter and matching closing delimiter are part of the string:

print(“Hello World !!")

Hello World !!

type("Hello World !!")

<class 'str'>

Escape Sequences in Strings

You can accomplish this using a backslash (\) character. A backslash character in a string indicates that one or more characters that follow it should be treated specially.

Suppressing Special Character Meaning

>>> print('This string contains a single quote (') character.')

SyntaxError: invalid syntax

Specifying a backslash in front of the quote character in a string “escapes” it and causes Python to suppress its usual special meaning. It is then interpreted simply as a literal single quote character:

print('This string contains a single quote (\') character.')

This string contains a single quote (') character.

Boolean Type

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False:

type(True)

<class 'bool'>

type(False)

<class 'bool'>

Lists

Python Lists are just like dynamically sized arrays, declared in other languages.

A list is a collection of things, enclosed in [ ] and separated by commas.

A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

Example

var = [‘Pandas’, ‘NumPy’, ‘Scikit Learn’]

print(var)

print(var[0])

Lists

Accessing elements from a multi-dimensional list

```
# Creating a Multi-Dimensional List
# (By Nesting a list inside a List)
List = [['Python', 'Java'], ['C']]
# accessing an element from the
# Multi-Dimensional List using
# index number
print("Accessing a element from a Multi-Dimensional list")
print(List[0][1])
print(List[1][0])
```

Negative indexing

In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

```
List = ['Python', 'Java', 'C', 'Scala']
print("Accessing element using negative indexing")
# print the last element of list
print(List[-1])
# print the third last element of list
print(List[-3])
```

Getting the size of Python list

Python len() is used to get the length of the list.

```
List = ['Python', 'Java', 'C', 'Scala']
print(len(List))
```

Adding Elements to a Python List

append() method:

Elements can be added to the List by using the built-in append() function. Only one element at a time can be added to the list by using the append() method, for the addition of multiple elements with the append() method, loops are used.

```
List = []
print("Initial blank List: ")
print(List)
 # Addition of Elements
# in the List
List.append(1)
```

Lists

Adding Elements to a Python List

insert() method:

append() method only works for the addition of elements at the end of the List, for the addition of elements at the desired position, insert() method is used. Unlike append() which takes only one argument, the insert() method requires two arguments(position, value).

```
# Creating a List
List = [1,2,3,4]
print("Initial List: ")
print(List)
 # Addition of Element at
# specific Position
# (using Insert Method)
List.insert(3, 12)
List.insert(0, 14)
print("\nList after performing Insert Operation: ")
print(List)
```

extend() method:

Other than append() and insert() methods, there's one more method for the Addition of elements, extend(), this method is used to add multiple elements at the same time at the end of the list.

append() and extend() methods can only add elements at the end.

```
List = [1, 2, 3, 4]
print("Initial List: ")
print(List)
# Addition of multiple elements
# to the List at the end
# (using Extend Method)
List.extend([8, '5', '6'])
print("\nList after performing Extend Operation: ")
print(List)
```

Lists

Removing Elements from the List

Using pop() method

pop() function can also be used to remove and return an element from the list, but by default it removes only the last element of the list, to remove an element from a specific position of the List, the index of the element is passed as an argument to the pop() method.

```
List = [1, 2, 3, 4, 5]
 # Removing element from the list using the pop() method
List.pop()
print("\nList after popping an element: ")
print(List)
# Removing element at a specific location from the list using the pop() method
List.pop(2)
print("\nList after popping a specific element: ")
print(List)
```

Slicing of a List

We can get substrings and sublists using a slice. To print a specific range of elements from the list, we use the Slice operation.

Slice operation is performed on Lists with the use of a colon(:).

To print elements from beginning to a range use:

[: Index]

To print elements from end-use:

[:-Index]

To print elements from a specific Index till the end use

[Index:]

To print the whole list in reverse order, use

[::-1]

Tuples

Tuple is a collection of Python objects much like a list. The sequence of values stored in a tuple can be of any type, and they are indexed by integers.

Tuple is immutable.

```
# Creating an empty Tuple
tup = ()
print("Initial empty Tuple: ")
print(tup)

---------------------------

# Creating a Tuple  with the use of string
tup = ('Python', 'Java')
print("\nTuple with the use of String: ")
print(tup)

# Creating a Tuple with the use of list
list1 = [1, 2, 4, 5, 6]
print("\nTuple using List: ")
print(tuple(list1))

----------------------
# Creating a Tuple
# with the use of built-in function
Tuple1 = tuple('Python')
print("\nTuple with the use of function: ")
print(Tuple1)
```

Dictionaries

Dictionary in Python is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.

```
dict = {1: ‘Python', 2: ‘Java', 3: ‘C'}
print(dict)
```

Creating a Dictionary

In Python, a dictionary can be created by placing a sequence of elements within curly {} braces, separated by ‘comma’. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any data type and can be duplicated, whereas keys can’t be repeated and must be immutable.

Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.

```
# Creating a Dictionary with Integer Keys
dict = {1: ‘Python', 2: ‘Java', 3: ‘C'}
print("\nDictionary with the use of Integer Keys: ")
print(dict)

# Creating a Dictionary with Mixed keys
dict = {'Name’: ‘Python', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(dict)

# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Creating a Dictionary with dict() method
Dict = dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})
print("\nDictionary with the use of dict(): ")
print(Dict)

# Creating a Dictionary with each item as a Pair
Dict = dict([(1, 'Geeks'), (2, 'For')])
print("\nDictionary with each item as a pair: ")
print(Dict)

# Creating a Nested Dictionary
Dict = {1: 'Geeks', 2: 'For',
    3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}

print(Dict
```

Adding elements to a Dictionary

One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = 'Value'.

Updating an existing value in a Dictionary can be done by using the built-in update() method.

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)
# Adding elements one at a time
Dict[0] = 'Python'
Dict[2] = 'Java'
print("\nDictionary after adding 2 elements: ")
print(Dict)
# Adding set of values to a single Key
Dict['Value_set'] = 2, 3, 4
print("\nDictionary after adding 3 elements: ")
print(Dict)
# Updating existing Key's Value
Dict[2] = 'Welcome'
print("\nUpdated key value: ")
print(Dict)
```

Accessing elements of a Dictionary

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets.

```
Dict = {1: 'Python', 2: 'Java', 3: 'Scala'}

# accessing a element using key
print("Accessing a element using key:")
print(Dict[1])
```

Sets

Set is an unordered collection of data types that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.

Creating a Set

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by a 'comma'.

A set cannot have mutable elements like a list or dictionary, as it is mutable.

Sets

```
# Creating a Set
my_set = set()
print('Initial blank Set: ')
print(my_set)
 # Creating a Set with the use of a String
my_set = set('hello world !!')
print("\nSet with the use of String: ")
print(my_set)
 # Creating a Set with the use of Constructo (Using object to Store String)
my_str = 'hello world !!'
my_set = set(my_str)
print("\nSet with the use of an Object: " )
print(my_set)
 # Creating a Set with the use of a List
my_set = set(["Python", "Java", "C"])
print("\nSet with the use of List: ")
print(my_set)
```

Arrays

An array is a collection of items stored at contiguous memory locations. It stores multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array.

Creating a Array

Array in Python can be created by importing array module. array(data_type, value_list) is used to create an array with data type and value list specified in its arguments.

```
# importing "array" for array creations
import array as arr
# creating an array with integer type
a = arr.array('i', [1, 2, 3])
# printing original array
print ("The new created array is : ", end =" ")
for i in range (0, 3):
    print (a[i], end =" ")
print()
# creating an array with float type
b = arr.array('d', [2.5, 3.2, 3.3])
# printing original array
print ("The new created array is : ", end =" ")
for i in range (0, 3):
    print (b[i], end =" ")
```

Python OOPs

Object Oriented Programming concept is basically using Objects and Classes in programming

The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

Concepts of Object-Oriented Programming (OOPs)

- **Class**
- **Objects**
- **Inheritance**
- **Polymorphism**
- **Encapsulation**
- **Data Abstraction**

Class

A **class** is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

- Classes are created by keyword **class**.
- **Attributes** are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute

Syntax

```
class ClassName:
    # Statement-1
    # Statement-N
```

```
class Vehicle:
    def __init__(self, brand, model, type):
        self.brand = brand
        self.model = model
        self.type = type
        self.gas_tank_size = 16
        self.fuel_level = 0

    def fuel_up(self):
        self.fuel_level = self.gas_tank_size
        print('Gas tank is now full.')

    def drive(self):
        print(f'The {self.model} is now driving.')
```

The self

Class methods must have an extra first parameter in the method definition. We do not give a value for this parameter when we call the method, Python provides it

If we have a method that takes no arguments, then we still have to have one argument.

When we call a method of this object as myobject.method(arg1, arg2), this is automatically converted by Python into MyClass.method(myobject, arg1, arg2)

The __init__ method

The __init__ method is similar to constructors in Java & C++. It is run as soon as an object of a class is instantiated.

The object is an entity that has a state and behaviour associated with it.

An object consists of :

State: It is represented by the attributes of an object. It also reflects the properties of an object.

Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.

Identity: It gives a unique name to an object and enables one object to interact with other objects.

Example: Creating an object

```
vehicle_obj = Vehicle('BMW', 'C Series', 'Truck')

print(vehicle_obj.brand)
print(vehicle_obj.model)
print(vehicle_obj.type)

vehicle_obj.fuel_up()
vehicle_obj.drive()
```

Class and Instance Variables

Instance variables are for data, unique to each instance and class variables are for attributes and methods shared by all instances of the class.

Instance variables are variables whose value is assigned inside a constructor or method with self whereas class variables are variables whose value is assigned in the class.

Python OOPs - inheritance

Inheritance

Inheritance is the capability of one class to derive or inherit the properties from another class. The class that derives properties is called the derived class or child class and the class from which the properties are being derived is called the base class or parent class.

Benefits:

- It provides the reusability of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.

- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Inheritance

Types of Inheritance –

Single Inheritance:

Single-level inheritance enables a derived class to inherit characteristics from a single-parent class.

Multilevel Inheritance:

Multi-level inheritance enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.

Hierarchical Inheritance:

Hierarchical level inheritance enables more than one derived class to inherit properties from a parent class.

Multiple Inheritance:

Multiple level inheritance enables one derived class to inherit properties from more than one base class.

Python OOPs - Polymorphism

Polymorphism in Python is the ability of an object to take many forms. In simple words, polymorphism allows us to perform the same action in many different ways.

Function Polymorphism in Python

There are some functions in Python which are compatible to run with multiple data types.

Example 2: Polymorphic len() function

```
print(len("Programiz"))
print(len(["Python", "Java", "C"]))
print(len({"Name": "John", "Address": "Nepal"}))
```

Python OOPs - inheritance

Inheritance

Inheritance is the capability of one class to derive or inherit the properties from another class. The class that derives properties is called the derived class or child class and the class from which the properties are being derived is called the base class or parent class.

Benefits:

- It provides the reusability of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.

- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Python OOPs - Encapsulation

Access Modifiers in Python

Encapsulation can be achieved by declaring the data members and methods of a class either as private or protected.
In Python, we don't have direct access modifiers like public, private, and protected.
We can achieve this by using single underscore and double underscores.

Access modifiers limit access to the variables and methods of a class. Python provides three types of access modifiers private, public, and protected.

- **Public Member**: Accessible anywhere from outside class.
- **Private Member**: Accessible within the class
- **Protected Member**: Accessible within the class and its sub-classes

class PetrolVehicle:

```
def __init__(self, brand, model, type):
    self.brand = brand
    self.model = model          →  Public Member (Accessible within or outside class)
    self._type = type           →  Protected Member (Accessible within the class and subclasses)
    self.__vehicle_type = 'Petrol'  →  Private Member (Accessible withing the class)
```

Python OOPs - Abstraction

Abstraction is used to hide the internal functionality of the function from the users.

The users only interact with the basic implementation of the function, but inner working is hidden.

User is familiar with that "what function does" but they don't know "how it does.“

In Python, Abstraction works by incorporating abstract classes and methods.

Abstract Class: A class specified in the code that has abstract methods is named Abstract Class.

Abstract Method: It doesn’t have any implementation. All the implementations are done inside the sub-classes.

Abstraction classes in Python

In Python, abstraction can be achieved by using abstract classes and interfaces.

- A class that consists of one or more abstract method is called the abstract class.
- Abstract methods do not contain their implementation.
- Abstract class can be inherited by the subclass and abstract method gets its definition in the subclass.
- Abstraction classes are meant to be the blueprint of the other class.
- An abstract class is also helpful to provide the standard interface for different implementations of components.
- Python provides the abc module to use the abstraction in the Python program. Let's see the following syntax.

```
from abc import ABC
class ClassName(ABC):
    pass
```

We import the ABC class from the abc module.

Abstract Base Classes

An abstract base class is the common application program of the interface for a set of subclasses.
It can be used by the third-party, which will provide the implementations such as with plugins.

Working of the Abstract Classes

Python doesn't provide the abstract class itself. We need to import the **abc** module, which provides the base for defining Abstract Base classes (ABC). The ABC works by decorating methods of the base class as abstract. It registers concrete classes as the implementation of the abstract base. We use the **@abstractmethod** decorator to define an abstract method or if we don't provide the definition to the method, it automatically becomes the abstract method

Syntax

```
from abc import ABC, abstractmethod
class Vehicle(ABC):
    def drive(self):
        pass
class ElectricVehicle(Vehicle):
    def drive(self):
        print("Driving Electric Vehicle")
class PetrolVehicle(Vehicle):
    def drive(self):
        print("Driving Petrol Vehicle")
#----------------------------------------
electric_vehicle= ElectricVehicle()
electric_vehicle.drive()
petrol_vehicle = PetrolVehicle()
petrol_vehicle.drive()
```

Below are the points which we should remember about the abstract base class in Python.

- An Abstract class can contain the both method normal and abstract method.
- An Abstract cannot be instantiated; we cannot create objects for the abstract class.
- The derived class implementation methods are defined in abstract base classes.