Python Scope:
A variable is only available from inside the region it is created. This is called **scope**.

Local Scope
A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

Global Scope
A variable created in the main body of the Python code is a global variable and belongs to the global scope.

Naming Variables
If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function)

Global Keyword
If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.
The global keyword makes the variable global.

In [ ]:
```
#Example
#A variable created inside a function is available inside that function:

def myfunc():
  x = 400
  print(x)

myfunc()
```

```
400
```

In [ ]:
```
#Example
#A variable created outside of a function is global and can be used by anyone:

x = 300 # global

def myfunc():
  x=200 #local
  print(x)
myfunc ( )

print(x)
```

```
200
300
```

In [ ]:
```
#Example
#To change the value of a global variable inside a function, refer to the variable by using the global keyword:

x = 300

def myfunc():
  global x
  x = 200
```

```
  print(x)
myfunc()

print(x)
```

```
200
200
```

Python Modules:

Consider a module to be the same as a code library.
A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension .py

Use a Module

Now we can use the module we just created, by using the import statement:

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc)

Re-naming a Module

You can create an alias when you import a module, by using the as keyword

Import From Module

You can choose to import only parts from a module, by using the from keyword.

In [ ]:
```
#from google.colab import drive
#drive.mount('/content/drive')

path = "/content/mymodule.py"
```

In [ ]:
```
#Example
#Save this code in a file named mymodule.py

def greeting(name):
  print("Hello, " + name)
```

In [ ]:
```
#Example
#Import the module named mymodule, and call the greeting function:

import mymodule

mymodule.greetngs("shikha")
```

Out [5]: 'shikha'

In [ ]:
```
import mymodule1 as my
my.thislist()
my.greetngs("shikha")
```

```
[1, 2, 3, 4]
```

Out [9]: 'shikha'

In [ ]:
```
#Example
#Save this code in the file mymodule.py

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

In [ ]:
```
#Example
#Import the module named mymodule, and access the person1 dictionary:

import mymodule2
a = mymodule2.person1["age"]
print(a)
```

```
36
```

In [ ]:
```
#Example
#Create an alias for mymodule called mx:

import mymodule2 as mx

a = mx.person1["age"]
print(a)
```

In [ ]:
```
#Example
#The module named mymodule has one function and one dictionary:

def greeting(name):
  print("Hello, " + name)

person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}

#Example
#Import only the person1 dictionary from the module:

from mymodule import thislist

mymodule.thislist()
mymodule.greeting("shikha")
#print (person1["name"])
```

```
[1, 2, 3, 4]
```

Out [7]: 'helloshikha'

Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Create a Parent Class

Any class can be a parent class, so the syntax is the same as creating any other class

In [ ]:
```
#Example
#Create a class named Person, with firstname and lastname properties, and a printname method:

class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("harshita", "pal")
x.printname()
```

```
harshita pal
```

Create a Child Class

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class

In [ ]:
```
#Example
#Create a class named Student, which will inherit the properties and methods from the Person class:

class Student(Person):
  pass
```

In [ ]:
```
#Example
#Use the Student class to create an object, and then execute the printname method:

class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  pass

x = Student("Mike", "Olsen")
x.printname()
```

Add the init() Function

The init() function is called automatically every time the class is being used to create a new object.

When you add the init() function, the child class will no longer inherit the parent's init() function.

The child's init() function overrides the inheritance of the parent's init() function.

To keep the inheritance of the parent's init() function, add a call to the parent's init() function

In [ ]:
```
#Example Add the __init__() function to the Student class:
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

In [ ]:
```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)


x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

Use the super() Function

Python also has a super() function that will make the child class inherit all the methods and properties from its parent

In [ ]:
```
#Example
class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
```

In [ ]:
```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)

x = Student("Mike", "Olsen")
x.printname()
```

```
Mike Olsen
```

Add Properties

In [ ]:
```
#Example
#Add a property called graduationyear to the Student class:

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019
```

In [ ]:
```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2019

x = Student("Mike", "Olsen")
x.printname()
print(x.graduationyear)
```

```
Mike Olsen
2019
```

In [ ]:
```
#Example
#Add a year parameter, and pass the correct year when creating objects:

class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Mike", "Olsen", 2019)
x.printname()
print(x.graduationyear)
```

```
Mike Olsen
2019
```

Add Methods

In [ ]:

```
#Example
#Add a method called welcome to the Student class:

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)
```

In [ ]:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

  def welcome(self):
    print("Welcome parent ", self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)

x = Student("Mike", "Olsen", 2019)
x.welcome()
```

```
Welcome Mike Olsen to the class of 2019
```

If you add a method in the child class with the same name as a function in the parent class, the inheritance of the parent method will be overridden.