```
# Tutorial Week 8: Inheritance in Python
# B. Schoen-Phelan
# 11-11-2020

# basic inheritance

# class A:
#   def __init__(self):
#     print("I'm class A")
#
# class B(A):
#   pass
#
# a = A()
# b = B()
#
# # usage of isinstance
# print(isinstance(21, int))
# print(isinstance(a,A))
# print(isinstance(b,A))
#
# # compare to type
# print(type(21))
# print(type(a))
# print(type(b))
# print(type(b)==A, type(b)==B)

# example from W3 schools

# class Person:
#   def __init__(self, f_name, l_name):
#     self.first_name = f_name
#     self.last_name = l_name
#
#   def print_name(self):
#     print(self.first_name, self.last_name)

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

```
# bsp = Person("Bianca", "Phelan")
# bsp.print_name()

# class Student(Person):
#   pass

# you have access to the methods and attributes
# of the parent class without having to
# program any of the functionality
# x = Student("John", "Doe")
# x.print_name()
# notice how we do not need an instance of Person
# in order to use the Student. The definition of
# Person was enough to be able to use it.

# now extend Student class with more functionality
# class Student(Person):
#   # init in Student overrides init from Person
#   def __init__(self, s_id, f_name, l_name):
#     self.student_id = s_id
#
#
# x = Student(12345, "John", "Doe")
# x.print_name() # causes an error if not defined in
init

# now corrected student:
# class Student(Person):
#   # init in Student overrides init from Person
#   # super() grabs the class one higher in the
hierarchy
#   # the parent class and initialises the fname and
lname
#   def __init__(self, s_id, f_name, l_name):
#     super().__init__(f_name, l_name)
#     self.student_id = s_id
#
#
```

```
# x = Student(12345, "John", "Doe")
# x.print_name()
# print(x.student_id)


# quick example about private and very private
variables in Python
# class A:
#     def __init__(self):
#         self.public = "Lecturer"
#         self._private = "Bianca"
#         self.__very_private = "Phelan"


# bianca = A()
# print(bianca.public)
# print(bianca._private)
# print(bianca.__very_private)  # this one causes an
error

# class A:
#   def __init__(self):
#     self.public = "Lecturer"
#     self._private = "Bianca"
#     self.__very_private = "Phelan"
#
#   @property
#   def very_private(self):
#     return self.__very_private
#
#   @very_private.setter
#   def very_private(self, value):
#     if type(value) == str:
#       self.__very_private = value
#     else:
#       raise Exception("Error, cannot set this
value.")
#
```

```
# bianca = A()
# print(bianca.very_private)
#
# try:
#   bianca.very_private = "Schoen" # switch this out
for a 3 or other non string values
# except Exception as e:
#   print("Tried setting non string", e)
# finally:
#   print("Stays a string")
#
# print(bianca.very_private)

# class B(A):
#   pass
#
# bryan = B()
# print(bryan.public)
# print(bryan._private)
# # print(bryan.__very_private) #causes an error
# print(bryan.very_private) # also available, like
before
# bryan.very_private = "Duggan"
# print(bryan.very_private)

# back to Person and Student

class Person:
 def __init__(self, f_name, l_name):
   self.__first_name = f_name
   self.__last_name = l_name

 @property
 def first_name(self):
   # include validation if needed, maybe we return
names
   # only on a Wednesday, or some business logic like
that
   return self.__first_name
```

```
 @property
 def last_name(self):
   return self.__last_name

 @last_name.setter
 def last_name(self, value):
   # imagine some validation here
   self.__last_name = value

 def print_name(self):
   print(self.first_name, self.last_name)


# bianca = Person("Bianca", "Phelan")
# bianca.print_name()
# bianca.first_name = "Susan" #causes an error because
it's not set
# bianca.last_name = "Schoen"
# bianca.print_name()

# class Student(Person):
#   def __init__(self, s_id, f_name, l_name):
#     super().__init__(f_name, l_name)
#     self.__student_id = s_id
#
#   @property
#   def get_student_id(self):
#     return self.__student_id
#
#   def get_full_details(self):
#     return super().first_name, super().last_name,
self.get_student_id
#
# bryan = Student(1234, "Bryan", "Duggan")
# print(bryan.get_full_details())
# print(bryan.get_student_id)
# # bryan.  #see what is available, nothing available
to change the student id
```

```
# # bryan.first_name = "Brian" #fails like before
# bryan.last_name = "Dug"
# print(bryan.get_full_details())

# more on inheritance:
# multiple inheritance
# example from digital ocean
#
https://www.digitalocean.com/community/tutorials/unders
tanding-class-inheritance-in-python-3
class Coral:

 def community(self):
   print("Coral lives in a community.")

 def same_name(self):
   print("hello from Coral.")


class Anemone:

 def protect_clownfish(self):
   print("The anemone is protecting the clownfish.")

 def same_name(self):
   print("hello from Anemone.")


class CoralReef(Coral, Anemone):
 pass

# great_barrier = CoralReef()
# great_barrier.community()
# great_barrier.protect_clownfish()
# great_barrier.same_name() #picks the one that was
mentioned first (MRO)

import inspect
```

```
print(inspect.getmro(CoralReef)) #list is dependent on
which one was named first
print(inspect.getmro(str)) #works for everything, see
how all inherit from object

x = object()
print(x.__class__)
# all classes we create are derived from object, even
if not explicitely said
class A (object):
 pass

a = A()
print(a.__class__)
print(inspect.getmro(A))
```