# Python

Object Oriented Programming: Classes, Objects, Inheritance

# In this lecture

- Object Oriented Programming Theory
- Classes and Objects in Python
  - 'self' reference
  - __init__() constructor
- Inheritance
  - Parent and Child terminology
  - 'super()' reference
  - Extending Classes

# Reminder!

# type() function

In[ ]:

```
age = 30
print(type(age))
name = "Nick"
print(type(name))
```

# type() function

In[ ]:

```
age = 30
print(type(age))
name = "Nick"
print(type(name))
```

```
<class 'int'>
<class 'str'>
```

# Classes and Objects

# OOP

- The Object-Oriented Paradigm originated in the 1980s.
- C++ was originally known as 'C with classes'.
- Procedural programming would separate data from procedures.
- Object Oriented Programming encapsulates both data and procedures into a package (an object).
- As we have seen, Python build primitive types as classes: int, str, float, bool

# Classes and Objects

Classes are the **blueprint**

**Unique objects** can be created from this blueprint

# Two parts

Variables
Functions

# Other terms!

Variables / data / properties / attributes

Functions / methods / behaviours / subroutines

# Classes and Objects in Python

# Classes and objects

In[ ]:

```
class Student:
    def print_name(self):
        print("Hi Nick")

obj = Student() # call constructor
obj.print_name()
```

# Classes and objects

In[ ]:

```
1 |  class Student:
2 |      def print_name(self):
3 |          print("Hi Nick")
4 |
5 |  obj = Student() # call constructor
6 |  obj.print_name()
```

# Classes and objects

In[ ]:

```
class Student:
    def print_name(self):
        print("Hi Nick")

obj = Student() # call constructor
obj.print_name()
```

# Classes and objects

In[ ]:

```
1 |  class Student:
2 |      def print_name(self):
3 |          print("Hi Nick")
4 |
5 |  obj = Student() # call constructor
6 |  obj.print_name()
```

# Classes and objects

In[ ]:

```
1 |  class Student:
2 |      def print_name(self):
3 |          print("Hi Nick")
4 |
5 |  obj = Student() # call constructor
6 |  obj.print_name()
```

# Classes and objects

In[ ]:

```
class Student:
    def print_name(self):
        print("Hi Nick")

obj = Student() # call constructor
obj.print_name()
```

```
Hi Nick
```

# Object addr

In[ ]:

```
1 |  class Student:
2 |      def print_name(self):
3 |          print("Hi Nick")
4 |
5 |  obj = Student() # call constructor
6 |  print(obj)
```

# Object addr

In[ ]:

```
1 |  class Student:
2 |     def print_name(self):
3 |         print("Hi Nick")
4 |
5 |  obj = Student() # call constructor
6 |  print(obj)
```

```
<__main__.Student at 0x1046a0be0>
```

# Two objects

In[ ]:

```
class Student:
    def print_name(self):
        print("Hi Nick")

nick = Student() # one object
print(nick)
sam = Student() # another object!
print(sam)
```

# Two objects

In[ ]:

```
1  class Student:
2      def print_name(self):
3          print("Hi Nick")
4
5  nick = Student() # one object
6  print(nick)
7  sam = Student() # another object!
8  print(sam)
```

```
<__main__.Student at 0x1046a0be0>
<__main__.Student at 0x10458bf40>
```

# Two objects

In[ ]:

```
class Student:
    def print_name(self):
        print("Hi Nick")

nick = Student() # one object
nick.print_name()
sam = Student() # another object!
sam.print_name()
```

# Problem!

In[ ]:

```
class Student:
    def print_name(self):
        print("Hi Nick")

nick = Student() # one object
nick.print_name()
sam = Student() # another object!
sam.print_name()
```

```
Hi Nick
Hi Nick
```

# 'self' reference

# Self

- 'self' can be substituted for any given object created of the class.
- Rather than specifying one object that will be referred to every time the method is run, 'self' can refer to the object that the method is being called on.
- In Python, self is automatically passed (so we don't have to), but it is received, so has to be defined in class methods.

# Self

In[ ]:

```
1 | class Student:
2 |     def set_name(self, name):
3 |         self.name = name
4 |
5 | nick = Student() # call constructor
6 | nick.set_name("Nick")
7 | print(nick.name)
```

# Self

In[ ]:

```
1 |  class Student:
2 |      def set_name(self, name):
3 |          self.name = name
4 |
5 |  nick = Student() # call constructor
6 |  nick.set_name("Nick")
7 |  print(nick.name)
```

# Self

In[ ]:

```
1 |  class Student:
2 |      def set_name(self, name):
3 |          self.name = name
4 |
5 |  nick = Student() # call constructor
6 |  nick.set_name("Nick")
7 |  print(nick.name)
```

# Self

In[ ]:

```
1 |  class Student:
2 |      def set_name(self, name):
3 |          self.name = name
4 |
5 |  nick = Student() # call constructor
6 |  nick.set_name("Nick")
7 |  print(nick.name)
```

# Self

In[ ]:

```
1 |  class Student:
2 |      def set_name(self, name):
3 |          self.name = name
4 |
5 |  nick = Student() # call constructor
6 |  nick.set_name("Nick")
7 |  print(nick.name)
```

# Self

In[ ]:

```
class Student:
    def set_name(self, name):
        self.name = name

nick = Student() # call constructor
nick.set_name("Nick")
print(nick.name)
```

```
Nick
```

# Self - unique values

In[ ]:

```
1  class Student:
2      def set_name(self, name):
3          self.name = name
4
5  nick = Student() # One object
6  nick.set_name("Nick")
7  sam = Student() # Another object
8  sam.set_name("Sam")
```

# __init__() constructor

# Constructor

- A constructor is a method which has the same name as the class.

- In Python, we can use the dunder method (**d**ouble **under**score) __init__() to refer to the constructor. Dunder methods are called by the **Python interpreter**. It **init**ialises the object and sets values for attributes (variables).
- The constructor is called when we create an object of the class.
- In Python we can still call the constructor (same name as the class). The Python interpreter then calls the **__init__() method**

# __init__()

In[ ]:

```
1 | class Student:
2 |     def __init__(self, name):
3 |         self.name = name
4 |
5 | nick = Student("Nick")
6 | print(nick.name)
```

# __init__()

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

nick = Student("Nick")
print(nick.name)
```

```
Nick
```

# Two objects

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

nick = Student("Nick")
print(nick.name)
sam = Student("Sam")
print(sam.name)
```

# Two objects

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

nick = Student("Nick")
print(nick.name)
sam = Student("Sam")
print(sam.name)
```

```
Nick
Sam
```

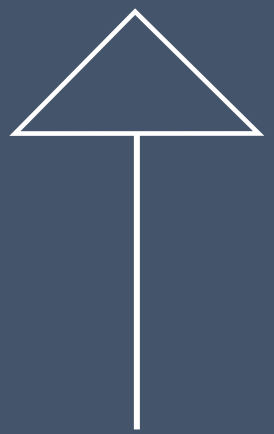
# Inheritance in Python

# Inheritance

- In our social world; inheritance means the passing down of assets from generation to generation; children inherit from their parents.
- In programming, inheritance is modelled by allowing 'child' classes to access variables and methods from the 'parent' class.
- Furthermore, Children classes often extend (are specialist versions of) the Parent classes.

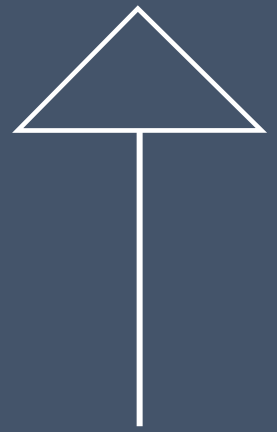
# Inheritance terms

Parent



Child

Base



Derived

Super

Sub

# Inheritance in C and Java

- Java:
  - ChildClass extends ParentClass
- C family languages use the : (colon) operator
  - ChildClass : ParentClass
- In Python, we have to pass the Parent reference to the Child class.

# Inheritance

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        print("Hi from Child Class")

childobj = Child() # call constructor
childobj.print()
```

# Inheritance

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        print("Hi from Child Class")

childobj = Child() # call constructor
childobj.print()
```

# Inheritance

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        print("Hi from Child Class")

childobj = Child() # call constructor
childobj.print()
```

# Inheritance

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        print("Hi from Child Class")

childobj = Child() # call constructor
childobj.print()
```

# Inheritance

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        print("Hi from Child Class")

childobj = Child() # call constructor
childobj.print()
```

# 'super()' reference

# Super

- We've used the keyword `self` within class constructors to refer to variables of any given object that is created.
- In an inheritance hierarchy, the child constructor may want to invoke the parent constructor to initialise values for inherited attributes.
- The `super()` is a reference to the parent's constructor.
- You can also refer to attributes and functions through the `super()` reference.

# super()

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        super().print()

obj = Child() # call constructor
obj.print()
```

# super()

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        super().print()

obj = Child() # call constructor
obj.print()
```

# super()

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        super().print()

obj = Child() # call constructor
obj.print()
```

# super()

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        super().print()

obj = Child() # call constructor
obj.print()
```

# super()

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        super().print()

obj = Child() # call constructor
obj.print()
```

# super()

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        super().print()

obj = Child() # call constructor
obj.print()
```

# super()

In[ ]:

```
class Parent:
    def print(self):
        print("Hi from Parent Class")

class Child(Parent):
    def print(self):
        super().print()

obj = Child() # call constructor
obj.print()
```

# Inheritance extension

# Inheritance

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

class PartTimeStudent(Student):
    def __init__(self, name):
        super().__init__(name)

nick = PartTimeStudent("Nick")
print(nick.name)
```

# Inheritance

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

class PartTimeStudent(Student):
    def __init__(self, name):
        super().__init__(name)

nick = PartTimeStudent("Nick")
print(nick.name)
```

# Inheritance

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

class PartTimeStudent(Student):
    def __init__(self, name):
        super().__init__(name)

nick = PartTimeStudent("Nick")
print(nick.name)
```

# Inheritance

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

class PartTimeStudent(Student):
    def __init__(self,name):
        super().__init__(name)

nick = PartTimeStudent("Nick")
print(nick.name)
```

# Inheritance

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

class PartTimeStudent(Student):
    def __init__(self, name, hours):
        super().__init__(name)
        self.hours = hours

nick = PartTimeStudent("Nick", 6)
```

# Inheritance

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

class PartTimeStudent(Student):
    def __init__(self, name, hours):
        super().__init__(name)
        self.hours = hours

nick = PartTimeStudent("Nick", 6)
```

# Inheritance

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

class PartTimeStudent(Student):
    def __init__(self, name, hours):
        super().__init__(name)
        self.hours = hours

nick = PartTimeStudent("Nick", 6)
```

# Inheritance

In[ ]:

```
class Student:
    def __init__(self, name):
        self.name = name

class PartTimeStudent(Student):
    def __init__(self, name, hours):
        super().__init__(name)
        self.hours = hours

nick = PartTimeStudent("Nick", 6)
```