Inheritance

Parents & Children

create parent class

```
## parent class
class [ParentClass]:
    def __init__(self, [parent_param1], [parent_param2]):
        self.[parent_attribute1] = [parent_param1]
        self.[parent_attribute2] = [parent_param2]

    def [parent_method](self):
        return [some_value]

    ...
```

create child class

```
## child class
class [ChildClass]([ParentClass]):
    def __init__(self, [parent_param1], [parent_param2], [child_param]):
        self.[parent_attribute1] = [parent_param1]
        self.[parent_attribute2] = [parent_param2]
        self.[child_attribute] = [child_param]

    ## automatically inherits all of parent's methods

    def [child_method](self): ## add child methods
        return [some_value]

    ...
```

`super()`

`super()` in `__init__`

```
## parent class
class [ParentClass]:
    def __init__(self, [parent_param1], [parent_param2]):
        self.[parent_attribute1] = [parent_param1]
        self.[parent_attribute2] = [parent_param2]

## child class (uses super() to inherit parents attribute handling)
class [ChildClass]([ParentClass]):
    def __init__(self, [parent_param1], [parent_param2], [child_param]):
        super().__init__([parent_param1], [parent_param2])
        self.[child_attribute] = [child_param]
```

`super()` in other methods

```
## parent class
class [ParentClass]:
    def [method_name](self):
        return [parent_value]

## child class
class [ChildClass]([ParentClass]):
    def [method_name](self):
        [parent_result] = super().[method_name]() ## call parent method
        ## add to/modify behaviour
        return [modified_value]
```

Learn more about the `super()` method here.

Polymorphism

"If It Fits, It Sits"

The same-looking method can work for multiple object types if defined (and/or yield different results).

```
## parent class
class [ParentClass]:
    def __init__(self, [param]):
        self.[attribute] = [param]

    def [method_name](self):
        return f"{self.[attribute]} [parent_behavior]"

## child class no. 1
class [ChildClass1]([ParentClass]):
    def [method_name](self):
        [parent_result] = super().[method_name]()
        return [parent_result] + "[child1_specific_behavior]"

## child class no. 2
class [ChildClass2]([ParentClass]):
    def [method_name](self):
        [parent_result] = super().[method_name]()
        return [parent_result] + "[child2_specific_behavior]"

[parent_object_instance].[method_name]() ## this will work
[child1_object_instance].[method_name]() ## this will work
[child2_object_instance].[method_name]() ## this will work too
```

Duck Typing

"If It Quacks Like a Duck, It's a Duck"

```
## note that the below classes are completely unrelated!
## class no. 1
class [Class1]:
    def [method_name](self):
        return "[class1_behavior]"

## class no. 2
class [Class2]:
    def [method_name](self):
        return "[class2_behavior]"

## class no. 3
class [Class3]:
    def [method_name](self):
        return "[class3_behavior]"

## function that uses duck typing
def [function_name]([obj]):
    return [obj].[method_name]()    ## if an object has such method, it will work

[function_name]([class1_instance])  ## this will work
[function_name]([class2_instance])  ## this will work
[function_name]([class3_instance])  ## this will work too
```

General Tips

1. **Confused about why use `super()` outside `__init__` when all methods get inherited anyway?**
   Use `super()` inside other methods when (i) extending the behaviour of a parent method, or (ii) overriding a parent method behaviour, but only partially.